Michael Weigend

Python

8. Auflage

GE-PACKT

- Schneller Zugriff auf Module, Klassen und Funktionen
- tkinter, Datenbanken, OOP und Internetprogrammierung
- Für die Version Python 3.8

mitp

mitp

Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Der Verlag räumt Ihnen mit dem Kauf des ebooks das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und Einspeicherung und Verarbeitung in elektronischen Systemen.

Der Verlag schützt seine ebooks vor Missbrauch des Urheberrechts durch ein digitales Rechtemanagement. Bei Kauf im Webshop des Verlages werden die ebooks mit einem nicht sichtbaren digitalen Wasserzeichen individuell pro Nutzer signiert.

Bei Kauf in anderen ebook-Webshops erfolgt die Signatur durch die Shopbetreiber. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

Michael Weigend

Python GE-PACKT

8. Auflage

mitp

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

ISBN 978-3-7475-0195-5
8. Auflage 2020

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Lektorat: Sabine Schulz
Sprachkorrektorat: Petra Heubach-Erdmann
Satz: III-satz, Husby, www.drei-satz.de

Inhaltsverzeichnis

E Einleitung 13

E.1 Was ist Python? 13
E.2 Einige besondere Merkmale von Python 13
E.3 Python 2 und 3 14
E.4 Hinweise zum Lesen dieses Buches 15

1 Basiskonzepte von Python 19

1.1 Python im interaktiven Modus 19
1.2 Ausführung von Python-Skripten 21
1.3 Die Zeilenstruktur 23
1.4 Deklaration der Codierung 26
1.5 Bezeichner (identifiers) 26
1.6 Objekte 28
1.7 Die Standard-Typ-Hierarchie 32
1.8 Literale für einfache Datentypen 33
1.9 Namensräume – lokale und globale Namen 39

2 Sequenzen 43

2.1 Gemeinsame Operationen für Sequenzen 43
2.2 Zeichenketten (Strings) 46
2.3 Tupel 52
2.4 Listen 53
2.5 Performance-Tipps 69

3 Dictionaries ... 73

4 Mengen ... 83

4.1 Der Typ set ... 83
4.2 Der Typ frozenset ... 84
4.3 Gemeinsame Operationen für set- und frozenset-Objekte ... 85
4.4 Mengen verändern ... 89

5 Operatoren ... 91

5.1 Unäre arithmetische Operatoren + - ~ ... 92
5.2 Binäre arithmetische Operatoren + - * / % ** ... 93
5.3 Bit-Operatoren << >> & ^ | ... 96
5.4 Vergleiche < <= > >= != == ... 98
5.5 Zugehörigkeit (in, not in) ... 100
5.6 Identitätsvergleich (is, is not) ... 101
5.7 Logische Operatoren (not, and, or) ... 102

6 Einfache Anweisungen (Statements) ... 105

7 Kontrollstrukturen ... 123

7.1 Verzweigungen – die if-Anweisung ... 123
7.2 Bedingte Ausdrücke ... 125
7.3 Verzweigungen mit logischen Operatoren ... 125
7.4 Iterationen – die for-Anweisung ... 127
7.5 Schleifen mit Abbruchbedingung – while ... 131
7.6 Abfangen von Laufzeitfehlern – try ... 133
7.7 Kontrollierte Ausführung – with ... 137

8 Definition von Funktionen ... 141

8.1 Aufruf und Ausführung einer Funktion ... 142
8.2 Funktionsnamen als Parameter ... 144
8.3 Voreingestellte Parameterwerte ... 145

8.4 Schlüsselwort-Argumente 146
8.5 Funktionen mit beliebiger Anzahl von Parametern ... 147
8.6 Funktionsannotationen: Typen zuordnen 148
8.7 Positionsargumente und Schlüsselwortargumente erzwingen (/, *) 150
8.8 Prozeduren 151
8.9 Rekursive Funktionen 152
8.10 Funktionen testen mit dem Profiler 152
8.11 Lokale Funktionen 154
8.12 Generatorfunktionen 155
8.13 Lambda-Formen 158
8.14 Decorators 159

9 Standardfunktionen (built in functions) und Standardtypen 163

10 Fehler und Ausnahmen 205

10.1 Syntaxfehler 205
10.2 Ausnahmen (Exceptions) 206
10.3 Erstellen einer eigenen Exception-Klasse 210
10.4 Testen von Vor- und Nachbedingungen mit assert ... 215
10.5 Selbstdokumentation im Debugging-Modus 216
10.6 Das Modul logging 218

11 Ein- und Ausgabe 229

11.1 Interaktive Eingabe über die Tastatur 229
11.2 Kommandozeilen-Argumente lesen 230
11.3 Formatierte Bildschirmausgabe 234
11.4 Lesbare Darstellung komplexer Objekte – das Modul pprint 237
11.5 Dateien 239
11.6 Objekte speichern – pickle 249
11.7 Zugriff auf beliebige Ressourcen über deren URL 255

12 Schnittstelle zum Laufzeitsystem – sys 257

13 Schnittstelle zum Betriebssystem – os und os.path 267

13.1 Das Modul os 267
13.2 Das Modul os.path 278

14 Datum und Zeit 285

14.1 Das Modul time 285
14.2 Das Modul datetime 291

15 Objektorientierte Programmierung mit Python 299

15.1 Definition von Klassen 300
15.2 Attribute 304
15.3 Methoden 307
15.4 Vererbung 317
15.5 Definition von Klassenbibliotheken 320

16 Verarbeitung von Zeichenketten 327

16.1 Standardmethoden für String-Objekte 327
16.2 Das Modul string 337
16.3 Formatierung mit dem %-Operator 340
16.4 Formatstrings 342
16.5 Reguläre Ausdrücke – das Modul re 347
16.6 Performance-Tipps zur Zeichenkettenbearbeitung . . . 359

17 Mathematische Funktionen 361

17.1 array 361
17.2 cmath 364
17.3 decimal 365
17.4 math 374
17.5 random 376

17.6 statistics 384
17.7 Das Modul secrets 388

18 CGI und WSGI 393

18.1 CGI-Skripte erstellen 393
18.2 Kommunikation über HTML-Formulare 398
18.3 Die Klasse cgi.FieldStorage 401
18.4 Das Modul cgitb – CGI-Skripte debuggen 406
18.5 Cookies 407
18.6 WSGI 410
18.7 Verarbeitung von Eingabedaten 412
18.8 mod_wscgi 414

19 Kommunikation im Internet 421

19.1 Das Modul ftplib 422
19.2 Erstellen eines CGI-Webservers 425
19.3 Das Modul imaplib 426
19.4 Das Modul poplib 428
19.5 Das Modul smtplib 431
19.6 Das Modul telnetlib 434

20 Datenbanken 437

20.1 Eine MySQL-Datenbank erstellen 438
20.2 Das Modul MySQLdb – Zugriff auf MySQL-Datenbanken 446
20.3 Das Modul sqlite3 451

21 Das Modul hashlib – Digitale Signaturen 455

21.1 Hashing-Objekte 456
21.2 Anwendung in der Sicherheitstechnik – Passwortgeschützte Online-Plattform 458

22 Grafische Benutzungsoberflächen 471

22.1 Widgets des Moduls tkinter 472
22.2 Die Benutzungsoberfläche als Aggregat von Widgets 473
22.3 Attribute der Widgets (Optionen) 476
22.4 Standard-Methoden der Widgets 486
22.5 Die Klasse Button 490
22.6 Die Klasse Canvas 491
22.7 Checkbutton 504
22.8 Entry 508
22.9 Frame 509
22.10 Label 510
22.11 Listbox 510
22.12 Menu 513
22.13 Menubutton 522
22.14 Die Klasse PhotoImage 525
22.15 Radiobutton 526
22.16 Scale 528
22.17 Scrollbar 531
22.18 Die Klasse Text 533
22.19 Tk 541
22.20 Layout-Manager 542
22.21 Kontrollvariablen 553
22.22 Dialogboxen 554
22.23 Event-Verarbeitung 556

23 Bild und Ton 565

23.1 Der Python Package Index 565
23.2 Die Python Imaging Library (PIL) 566
23.3 Klänge mit Winsound 583
23.4 PlaySound() 585

24 Threads 587

24.1 Funktionen in einem Thread ausführen: start_new_thread() 588
24.2 Thread-Objekte erzeugen – die Klasse Thread 589
24.3 Die Klasse Timer 592

25 XML 595

25.1 Das Modul xml.dom.minidom 596
25.2 Verarbeitung eines XML-Objektes – Einführendes Beispiel 597
25.3 Parsing – ein DOM-Objekt erstellen 600
25.4 Knoten eines DOM-Objektes – die Basisklasse Node 601
25.5 Die Klasse Document 610
25.6 Die Klasse Element 611
25.7 Die Klasse Text 615

A Ressourcen im Internet 617

A.1 Usenet 617
A.2 Mailinglisten 617
A.3 WWW 618

B Entwicklungsumgebungen 619

C Python-Module 621

D Von Python 2 zu Python 3 625

D.1 Unterschiede zwischen Python 2 und Python 3 625

E Glossar 629

Stichwortverzeichnis 641

E Einleitung

E.1 Was ist Python?

Python ist eine portable, interpretative, objektorientierte Programmiersprache. Ihre Entwicklung wurde 1989 von Guido van Rossum am Centrum voor Wiskunde en Informatica (CWI) in Amsterdam begonnen und wird nun durch die nichtkommerzielle Organisation »Python Software Foundation« (PSF, `http://www.python.org/psf`) koordiniert. Der Name soll an die britische Comedy-Gruppe Monty Python erinnern.

Ein Python-Programm – man bezeichnet es als Skript – ist ein Text, der von einem Interpreter ausgeführt werden kann. Weil Python-Skripte auf verschiedenen Systemplattformen (Unix, Windows, Mac OS) laufen können, bezeichnet man die Sprache als portabel. Die aktuelle Version von Python, auf die sich dieses Buch bezieht, ist Version 3.8. Sie kann von der Python-Homepage `http://www.python.org` heruntergeladen werden. Python ist kostenlos und kompatibel mit der GNU General Public License (GPL).

E.2 Einige besondere Merkmale von Python

- Die Python-Syntax ermöglicht sehr kompakte Programmtexte.
- Das Layout des Quelltextes dient nicht allein der besseren Lesbarkeit, sondern hat eine Bedeutung. So markiert das Zeilenende das Ende einer Anweisung. Anweisungsblöcke (wie z.B. das Innere einer Schleife) werden durch Einrückung festgelegt. Zeilen des Programmtextes, die

um die gleiche Anzahl von Stellen eingerückt sind, gehören zum gleichen Anweisungsblock.

- Mit Python kann man objektorientiert, imperativ und funktional programmieren.
- Im Unterschied etwa zu Pascal oder Java muss den Variablen kein Datentyp explizit zugeordnet werden. Es gibt keine Variablendeklarationen. Der Datentyp ergibt sich aus dem Kontext. Wenn es erforderlich ist, finden automatische Typkonvertierungen statt.
- Mehrere Variablen können zu Tupeln zusammengefasst werden, die in einer einzigen Zuweisung verarbeitet werden können. Die Komponenten eines Tupels werden durch Kommata getrennt. Durch diesen Mechanismus können Hilfsvariablen eingespart werden. So werden mit einer einzigen Anweisung `x,y = y,x` die Inhalte der Variablen `x` und `y` vertauscht.
- In der Mathematik übliche Schreibweisen wurden in die Syntax übernommen. Ausdrücke wie `a < b < c` oder verkettete Zuweisungen `a = b = c` sind erlaubt.
- Python verwendet wenige, aber sehr mächtige Sprachkonstrukte. Auf alles Überflüssige wurde verzichtet. In dieser Hinsicht kann man Python als »minimalistisch« bezeichnen.
- Objekte besitzen einen Wahrheitswert. So haben alle Zahlen ungleich null und alle nicht leeren Zeichenketten den Wahrheitswert »wahr«.
- Langzahl-Arithmetik (Rechnen mit ganzen Zahlen beliebiger Länge) ist integriert.

E.3 Python 2 und 3

Es ist wichtig zu wissen, dass es zwei unterschiedliche Sprachvarianten von Python gibt: Python 2 und Python 3. Im Jahre 2007 erschien mit Python 3 erstmals eine Version, die mit den Vorgänger-Versionen nicht kompatibel ist. Man entschied sich zu diesem Bruch, um einige grundlegende Designschwächen von Python zu beseitigen. Python wurde noch

konsistenter, als es bereits war. Wie bei Java können nun sämtliche Unicode-Zeichen für Bezeichner und Strings verwendet werden. Man unterscheidet nun zwischen Strings als Zeichenketten und Bytestrings als Oktettenfolgen. Es gibt keine `print`-Anweisung mehr, sondern eine Funktion `print()`.

Ältere Python-2-Programme können in der Regel nicht von einem Python-3-Interpreter ausgeführt werden. Die letzte Python 2-Version ist Python 2.7. Man kann es immer noch herunterladen und auf vielen Computern findet man noch Python 2. Aber seit 2020 wird Python 2 nicht mehr weiterentwickelt und gepflegt.

Dieses Buch verwendet durchgehend die Syntax von Python 3.

E.4 Hinweise zum Lesen dieses Buches

Typographie

- Python-Quelltext, Funktions- und Variablennamen, Operatoren, Grammatikregeln, Zahlen und mathematische Ausdrücke werden in einem Zeichenformat mit fester Breite gesetzt. Beispiele:

```
x = y + 1
(x < y) and (len(liste) < 5)
```

- Bei der Darstellung des Formats eines Funktionsaufrufs sind die Argumente kursiv. Sie sind – im Unterschied zum Funktionsnamen – Metabezeichner, die nicht Buchstabe für Buchstabe so aufgeschrieben werden, wie es angegeben ist, sondern durch andere Bezeichner oder Literale ersetzt werden können. Beispiel:

`range(`*zahl*`)`

- Wichtige Passagen in Python-Listings, auf die im Text Bezug genommen wird, sind zuweilen fett gedruckt, um das Wiederfinden zu erleichtern.

Beispiele

Die Beispiele mit Python-Quelltext in diesem Buch kann man in drei Gruppen einteilen:

- Auszüge aus einer interaktiven Session erkennt man an dem Python-Prompt >>>. Diese Beispiele kann man im Shell-Fenster einer Python-Entwicklungsumgebung (z.B. IDLE) ausprobieren. Hinter dem Prompt ist die Benutzereingabe. Zeilen ohne Prompt enthalten eine System-Antwort. Beispiel:

```
>>> x = 1
>>> y = 2
>>> print(x + y)
3
```

- Beispiele für eigenständige Python-Skripte oder Module mit Funktionsdefinitionen enthalten überhaupt kein Prompt. Sie werden in einem Editorfenster erstellt und dann durch Aufruf des Interpreters oder nach Import in einer interaktiven Session getestet. Beispiel:

```
# Quicksort
def qsort(L):
   if len(L) <= 1: return L
   else:
      return qsort( [ x for x in L[1:] if x < L[0]])\
           + [L[0]]  \
           + qsort( [y for y in L[1:] if y >= L[0]] )
```

- Für Aufrufe in einem Konsolenfenster des Betriebssystems (z.B. Unix-Shell oder MS-DOS-Eingabeaufforderung) verwenden wir das Prompt >. Beispiel:

```
> python addiere.py 1 27
28
```

Hinweise zum Aufbau der Kapitel

- Am Ende eines Unterkapitels mit einer Funktionsbeschreibung finden sich häufig Verweise auf andere Kapitel mit korrespondierendem Inhalt (»Siehe auch: ...«)
- Kapitel, in denen Module beschrieben werden, sind meist nach folgendem Schema aufgebaut:
  - Kurze Einleitung
  - Tabellarische Übersicht über die Objekte (Funktionen, Konstanten, Klassen, Methoden) des Moduls in alphabetischer Reihenfolge
  - Ausführliche Erklärung der wichtigsten Objekte mit Beispielen
  - Am Ende gelegentlich komplexere Anwendungsbeispiele
- Programmiertipps und wichtige Hinweise befinden sich in grauen Kästen.

1 Basiskonzepte von Python

1.1 Python im interaktiven Modus

Der Python-Interpreter kann in einem interaktiven Modus verwendet werden, in dem Sie einzelne Zeilen Programmtext eingeben und die Wirkung sofort beobachten können. Im interaktiven Modus können Sie mit Python experimentieren, etwa um sich in neue Programmiertechniken einzuarbeiten oder logische Fehler in einem Programm, das Sie gerade bearbeiten, aufzuspüren.

Der interaktive Python-Interpreter – die Python-Shell – kann auf verschiedene Weise gestartet werden:

1. In einem Konsolenfenster (z.B. Eingabeaufforderung bei Windows-Systemen) geben Sie das Kommando `python` ein. Damit das Betriebssystem den Python-Interpreter findet, muss der Systempfad richtig gesetzt sein. Bei einem Windows-System achten Sie bei der Installation von Python 3.8 darauf, dass auf der ersten Seite des Installationsprogramms unten die Checkbox ADD PYTHON 3.8 TO PATH gesetzt ist. Sie können natürlich auch nachträglich noch den Pfad setzen. Unter Windows 10 geben Sie in das Suchfeld links unten den Begriff `Systemeinstellungen` ein und drücken ENTER. Es erscheint eine Dialogbox mit mehreren Registerkarten. Wählen Sie die Registerkarte ERWEITERT und klicken Sie auf die Schaltfläche UMGEBUNGSVARIABLEN. Auf der nächsten Dialogseite wählen Sie die Variable PATH und klicken auf BEARBEITEN. Jetzt können Sie weitere Pfade hinzufügen.
2. Bei Windows-Rechnern klicken Sie auf den START-Button und klicken in der Liste Ihrer Apps im Ordner `Python` auf `Python3.8`. Es öffnet sich

ein Konsolenfenster (Eingabeaufforderung), in dem der Python-Interpreter läuft.

3. Sie öffnen die Entwicklungsumgebung IDLE, die zum Standardpaket gehört. Sie enthält neben einem Editorfenster ein eigenes Shell-Fenster, in dem man auf der Kommandozeile Python-Statements eingeben kann.

Die Python-Shell meldet sich immer mit einer kurzen Information über die Version und einigen weiteren Hinweisen. Dann folgt der charakteristische Promptstring aus drei spitzen Klammern >>> als Eingabeaufforderung. Hinter dem Prompt können Sie eine Anweisung eingeben und durch ↵ beenden. In den nächsten Zeilen kommt entweder eine Fehlermeldung, ein Funktionsergebnis oder (z.B. bei Zuweisungen) *keine* Systemantwort. Beispiel:

```
>>> 2+2
4
```

Wichtige Tastenkombinationen

Es lohnt sich, einige wenige Tastenkombinationen zur effizienten Bedienung der Python-Shell auswendig zu lernen:

Vorige Anweisung. Mit der Tastenkombination Alt + p können Sie den vorigen Befehl (previous command) noch einmal in die Kommandozeile schreiben. Drücken Sie mehrmals diese Tastenkombination, werden die noch weiter zurückliegenden Anweisungen geholt.

Nächste Anweisung. Wenn Sie mehrmals auf Alt + p gedrückt haben, können Sie mit Alt + N wieder zum nächstneueren Kommando springen (next command).

Keyboard Interrupt. Mit der Tastenkombination Strg + C können Sie den Abbruch eines laufenden Programms erzwingen. Das ist z.B. für das Testen von Programmen mit `while`-Anweisungen wichtig, weil eventuell eine Endlosschleife vorliegt und das Programm von alleine nicht anhält.

1.2 Ausführung von Python-Skripten

Python-Programme – meist nennt man sie Skripte – sind Textdateien, die unter einem Namen mit der Extension `.py` oder unter Windows auch `.pyw` abgespeichert werden, z.B. `hello.py`. Ein Python-Skript wird von einem Python-Interpreter ausgeführt (interpretiert), der letztlich den Programmtext in maschinenbezogene Befehle überführt. Das heißt: Das Skript ist plattformunabhängig, aber für jedes Betriebssystem gibt es einen eigenen Interpreter. Um ein Python-Skript ausführen zu können, muss dem Betriebssystem auf irgendeine Weise mitgeteilt werden, welches Programm es zur Interpretation einsetzen soll. Voraussetzung für die Ausführung eines Skripts ist, dass Python installiert und Systempfade, Umgebungsvariablen usw. korrekt gesetzt sind.

Grundsätzlich kann ein Python-Skript von einer Entwicklungsumgebung (z.B. IDLE) aus gestartet werden. Darüber hinaus gibt es folgende plattformabhängigen Möglichkeiten.

Windows

Unter Windows können Sie ein Python-Skript auf zweierlei Weise ausführen:

1. Öffnen Sie ein Konsolenfenster (Eingabeaufforderung) und geben Sie das Kommando `python` gefolgt vom Pfad des Python-Skripts ein. Beispiel:

   ```
   python meinSkript.py
   ```

2. Klicken Sie im Explorer-Fenster auf das Icon des Python-Skripts. Das Betriebssystem öffnet ein Konsolenfenster, in dem Ein- und Ausgaben erfolgen. Das funktioniert natürlich nur, wenn Programmnamen mit der Extension `.py` auch mit dem Python-Interpreter verbunden sind. Ist das nicht der Fall, klicken Sie das Programm-Icon mit der rechten Maustaste an und wählen im Kontextmenü die Option EIGENSCHAFTEN. Stellen Sie hinter ÖFFNEN MIT: den Verknüpfungspfad so ein, dass die Datei mit `python.exe` geöffnet wird. Nach Beendigung des

Programms wird das Fenster sofort wieder geschlossen. Das hat den Nachteil, dass die letzte Ausgabe des Programms nicht mehr gelesen werden kann. (Abhilfe bietet hier z.B. eine `input()`-Anweisung am Ende des Programms. Sie erzwingt, dass das DOS-Fenster geöffnet bleibt, bis die ↵-Taste gedrückt ist.)

Bei Programmen mit grafischer Benutzungsoberfläche, die in einem eigenen Anwendungsfenster laufen, wird es als störend empfunden, wenn sich zuerst ein Konsolenfenster öffnet. Solche Skripte sollte man unter einem Namen mit der Extension `.pyw` abspeichern, z.B. `editor.pyw`. Dann erscheint nach dem Anklicken des Programmicons kein DOS-Fenster, sondern sofort die Applikation.

Unix

Unter Unix kann man (wie bei MS Windows) in einem Shell-Fenster (Konsole) den Python-Interpreter durch ein Kommando der folgenden Form starten:

```
python meinSkript.py
```

Außerdem gibt es bei Unix den Mechanismus der so genannten *magic line*. In der ersten Zeile des Skripts kann spezifiziert werden, mit welchem Interpreter das Skript ausgeführt werden soll. Die magic line beginnt mit der Zeichenfolge #!, dahinter folgt entweder direkt der Pfad zum Python-Interpreter oder der Pfad zu einem Dictionary (`env`), in dem die Systemadministration den Pfad zum Python-Interpreter eingetragen hat. Wenn das Unix-System standardmäßig eingerichtet ist, müsste eine der beiden folgenden magic lines funktionieren:

```
#!/usr/bin/python
#!/usr/bin/env python
```

Das Python-Skript mit einer magic line ist direkt ausführbar. Zum Start reicht z.B. die Eingabe des Dateinamens in der Konsole. Voraussetzung ist allerdings, dass die Zugriffsrechte entsprechend gesetzt sind, das

heißt, das `executable`-Bit (x) muss mit Hilfe des Unix-Kommandos `chmod` auf `1` gesetzt sein.

CGI-Skripte, die von jedermann über das Internet gestartet werden können, müssen eine magic line enthalten. Die Rechtevergabe erfolgt üblicherweise nach folgendem Muster:

```
chmod 711 meinSkript.py
```

Damit hat der Besitzer alle Rechte, die anderen dürfen die Datei nur ausführen, nicht aber lesen oder ändern.

1.3 Die Zeilenstruktur

Ein Python-Skript ist eine Folge von Anweisungen. Im Unterschied zu anderen Programmiersprachen muss bei Python eine Anweisung nicht durch ein besonderes Zeichen (wie das Semikolon bei Java) abgeschlossen werden. Ebenso gibt es keine Zeichen für Beginn und Ende eines Anweisungsblocks (wie z.B. geschweifte Klammern in Java oder `begin` und `end` in Pascal).

Das Ende einer Anweisung wird durch das Zeilenende markiert. Somit darf sich eine Anweisung nicht über mehrere Zeilen erstrecken.

Erlaubt ist:

```
summe = 1 + 2
```

Nicht erlaubt ist:

```
summe = 1
+ 2
```

Python unterscheidet aber zwischen »physischen« und »logischen« Zeilen. Eine physische Zeile endet mit einem (unsichtbaren) betriebssystemabhängigen Steuerungssymbol für den Zeilenwechsel. Bei Unix ist das das ASCII-Zeichen LF (linefeed), bei DOS/Windows-Systemen die ASCII-Zeichenfolge CR LF (carriage return und linefeed) und bei Mac OS das ASCII-Zeichen CR.

Explizites Verbinden von Zeilen

Mit Hilfe eines Backslashs \ kann man in einem Python-Skript mehrere physische Zeilen zu einer logischen Zeile verbinden. Damit ist folgender Programmtext eine gültige Anweisung:

```
summe = 1 \
+ 2
```

Hinter dem Backslash darf aber in derselben Zeile kein Kommentarzeichen # stehen, denn ein Kommentarzeichen beendet eine logische Zeile. Nicht erlaubt ist also:

```
summe = 1 \ # Summenberechnung
+ 2
```

Implizites Verbinden von Zeilen

Geklammerte Ausdrücke (mit normalen, eckigen oder geschweiften Klammern) sind häufig sehr lang. Sie dürfen bei Python auf mehrere physische Zeilen verteilt werden und werden implizit zu einer einzigen logischen Zeile verbunden. Beispiele:

```
wochentage = ["Sonntag","Montag","Dienstag",
"Mittwoch", "Donnerstag", "Freitag", "Samstag"]
def volumen(h,    # Höhe,
            b,    # Breite und
            t):   # Tiefe eines Quaders
return h*b*t
```

Das zweite Beispiel zeigt auch Folgendes: Im Unterschied zu Zeilen, die mit einem Backslash \ explizit verbunden sind, dürfen implizit verbundene Zeilen Kommentare enthalten. Das ist auch sehr sinnvoll. Denn die Parameter einer Funktion möchte man häufig einzeln kommentieren.

Einrückungen – Anweisungsblöcke

Ein Anweisungsblock (in der Python-Dokumentation *suite* genannt) ist eine Folge von zusammengehörigen Anweisungen, z.B. das Innere einer Schleife, der Körper einer Funktionsdefinition oder ein Zweig einer if-else-Anweisung. Ein Block kann weitere Blöcke als Unterblöcke enthal-

ten. Beginn und Ende eines Blocks werden in einem Python-Skript nicht durch lesbare Symbole (geschweifte Klammern { } bei C oder Java), sondern durch eine Einrückung (indent) um eine bestimmte Anzahl von Stellen festgelegt. Beispiel:

```
a = 0
for i in range(5):
      a = a + i          # Beginn eines Blocks
      print(a)           # Ende des Blocks
print("Ende der Rechnung")
```

Die Anzahl der Leerzeichen vor dem ersten Nichtleerzeichen (Einrückungsgrad) ist beliebig. Wichtig ist allein, dass alle zusammengehörigen Anweisungen eines Blocks um exakt die gleiche Anzahl von Stellen eingerückt sind. Es ist gleichgültig, ob Sie beim Editieren Tabulatorzeichen (Tab) oder Leerzeichen verwenden. Empfohlen wird, Tabs und Leerzeichen nicht zu mischen. Der Python-Interpreter wandelt intern Tabulatorzeichen in die entsprechende Anzahl von Leerzeichen um. Am Ende eines Skripts werden alle begonnenen Blöcke vom Interpreter wieder beendet.

Das folgende Beispielskript zeigt einige typische Einrückungsfehler:

```
   def primzahl(n):                          #1
    teiler_gefunden = 0
    for i in range(2,n):
    if n%i == 0:                             #2
           print("keine Primzahl")
           teiler_gefunden = 1
            break                            #3
  if not teiler_gefunden:                    #4
           print("Primzahl")
```

#1: Fehler: Die erste Zeile darf nicht eingerückt sein.

#2: Fehler: Nach dem Doppelpunkt beginnt ein neuer Block, es muss eingerückt werden.

#3: Fehler: Unerwartete Einrückung. Die Zeile muss genauso weit eingerückt sein wie die Zeile davor.

#4: Fehler: Inkonsistente Einrückung. Die Anweisung gehört zum gleichen Block wie die `for`-Anweisung und muss ebenso weit eingerückt sein.

1.4 Deklaration der Codierung

Python-Skripte können in der ersten oder zweiten Zeile eine Zeile der Form

```
# -*- coding: <encoding-name> -*-
```

enthalten, die die Codierung des Textdokumentes angibt. Wenn die Deklaration der Codierung fehlt, wird utf-8 angenommen. Beispiele:

```
# -*- coding: latin-1 -*-
import sys, os
…
#!/usr/bin/python
# -*- coding: iso-8859-15 -*-
import sys, os
...
```

Siehe auch: Kapitel 16 (String-Methoden `decode()` und `encode()`)

1.5 Bezeichner (identifiers)

Bezeichner (identifiers) sind Zeichenketten, die für die Namen von Funktionen, Klassen, Variablen usw. verwendet werden dürfen. Ein Bezeichner (häufig spricht man auch von Namen) kann beliebig lang sein, muss mit einem Buchstaben oder einem Unterstrich beginnen und ist ansonsten aus Buchstaben, Ziffern und Unterstrichen zusammengesetzt. Es wird zwischen Groß- und Kleinschreibung unterschieden. Im Unterschied zu den Vorgängerversionen lässt Python 3 auch Unicode-Zeichen zu, die Buchstaben repräsentieren, aber nicht zum ASCII-Zeichensatz gehören. Erlaubt sind also insbesondere auch deutsche Umlaute und ß. Gültige Bezeichner sind z.B. `x`, `x1`, `straßenname`, `__privat`, `Class_1`.

Empfohlen wird allerdings häufig, für Bezeichner ASCII-Zeichen zu verwenden.

Schlüsselwörter (keywords)

Die folgenden Bezeichner sind reservierte Wörter oder Schlüsselwörter von Python. Sie dürfen nicht als Bezeichner z.B. für Variablennamen verwendet werden.

```
and         as        assert      break       class
continue    def       del         elif        else
except      False     finally     for         from
global      if        import      in          is
lambda      None      nonlocal    not         or
pass        raise     return      True        try
while       with      yield
```

Im Modul `keyword` gibt es eine Funktion, mit der getestet werden kann, ob ein String ein Schlüsselwort ist.

```
>>> import keyword
>>> keyword.iskeyword("with")
True
```

Verwendung von Unterstrichen

Unterstriche am Anfang und am Ende eines Bezeichners haben eine besondere Bedeutung.

Doppelte Unterstriche am Anfang und am Ende eines Bezeichners (z.B. `__init__`) kennzeichnen Objekte, die mit einem bestimmten »magischen« Verhalten verbunden sind. In Klassen z.B. gibt es immer eine Methode `__init__()`, die für die Initialisierung einer Instanz der Klasse zuständig ist (Konstruktor). Diese Methode wird aber nicht mit `__init__()` aufgerufen, sondern über den Namen der Klasse. Bezeichner mit doppelten Unterstrichen vorne und hinten verwendet man auch, um Operatoren zu überladen (siehe Kapitel 15.3).

Bezeichner, die mit zwei Unterstrichen beginnen, aber nicht mit Unterstrichen enden, werden für private Methoden und Attribute in Klassendefinitionen verwendet.

Ein einzelner Unterstrich am Anfang (z.B. `_irgendwas`) bewirkt, dass das bezeichnete Objekt durch die Anweisung `from` *`modul`* `import *` nicht in den Namensraum importiert wird. Bezeichner, die mit einem Unterstrich beginnen, sind also für interne Attribute und Methoden gedacht, die nur innerhalb eines Moduls verwendet werden.

Einen einzelnen Unterstrich am Ende verwendet man üblicherweise, um Namenskonflikte mit existierenden Schlüsselwörtern zu vermeiden.

Siehe auch: Kapitel 15.3

1.6 Objekte

Identität von Objekten

Python ist in einem sehr umfassenden Sinne eine objektorientierte Programmiersprache. Daten, Funktionen, Programmcode, Ausnahmen, Prozessframes und andere Sprachelemente werden durch Objekte repräsentiert. Jedes Objekt besitzt eine Identität, einen Wert und einen Typ.

Die Identität eines Objektes wird durch eine (einmalige) ganze Zahl repräsentiert, die mit der Standardfunktion `id()` abgefragt werden kann. Zwei Objekte mit gleichem Wert können unterschiedliche Identität besitzen. Mit dem Operator `is` kann ermittelt werden, ob zwei Objekte (angesprochen über Namen) dieselbe Identität besitzen.

Änderbare und unveränderbare Objekte

Bei manchen Objekten kann sich der Wert während ihrer Lebensdauer ändern. Man bezeichnet sie als änderbar (mutable). Zu den änderbaren Objekten gehören Listen. Objekte, die bei ihrer Erschaffung einen festen Wert erhalten, den sie bis zu ihrer Zerstörung behalten, bezeichnet man als unveränderbar (immutable). Zu den unveränderbaren Objekten gehö-