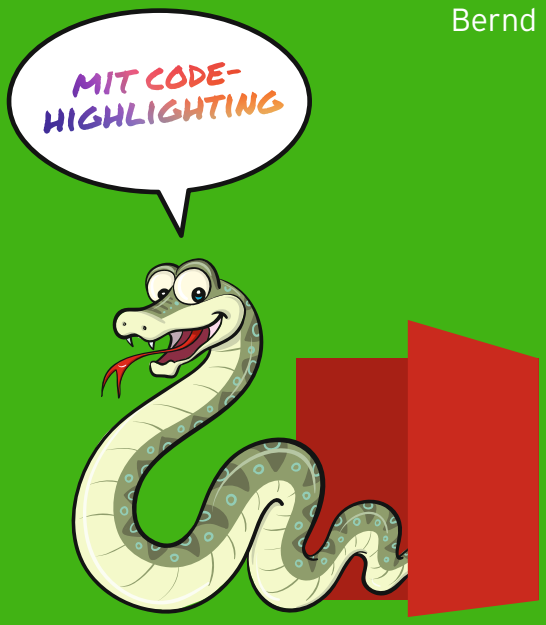
Bernd KLEIN



EINFÜHRUNG IN PYTHON 3

FÜR EIN- UND UMSTEIGER

4. Auflage

Im Internet:
Musterlösungen zu den Übungen

HANSER

Klein

Einführung in Python 3

Bernd Klein

Einführung in Python 3

Für Ein- und Umsteiger

4., vollständig überarbeitete Auflage

HANSER

Der Autor:
Bernd Klein, bernd@python-kurs.eu

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.

Lektorat: Brigitte Bauer-Schiewek
Copy editing: Jürgen Dubau, Freiburg/Elbe
Layout: le-tex publishing services, Leipzig
Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München
Umschlagrealisation: Max Kostopoulos
Titelmotiv: © istockphoto.com/zaricm
Druck und Bindung: Eberl & Kœsel GmbH & Co. KG, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-46379-0
E-Book-ISBN: 978-3-446-46556-5
E-Pub-ISBN: 978-3-446-46467-4

Inhalt

Vorwort

Ist es wirklich so, dass Vorworte – ähnlich wie Bedienungsanleitungen – meistens nicht gelesen werden? Auch wenn dies sicherlich für viele zutreffen mag, so gibt es Situationen, in denen gerade ein Vorwort wertvolle Dienste leisten kann. Zum Beispiel, um die Kaufentscheidung für ein Buch zu erleichtern. So stehen auch Sie jetzt vielleicht am Regal einer guten Buchhandlung und werden möglicherweise von zwei Fragen bewegt: Sie brauchen noch eine Bestätigung, dass Python die richtige Programmiersprache für Sie ist, und möchten wissen, wie Ihnen dieses Buch helfen wird, die Sprache schnell und effizient zu erlernen.

Für Python spricht der traumhafte Anstieg seiner Bedeutung in Wissenschaft, Forschung und Wirtschaft in den letzten Jahren. Dies spiegelt sich auch in den Rankings von Programmiersprachen wider, die von verschiedenen Stellen durchgeführt werden. PYPL zählt wohl zu den bekanntesten und anerkanntesten unter diesen Webrankings. Python führt im Dezember 2020 mit 30,34 % den Index von PYPL an, gefolgt von Java mit nur 17,23 %.[1]

Weitere wichtige Gründe, Python zu lernen, sind: Python ist mit seiner einfachen natürlichen Syntax sehr einfach zu lernen. Python läuft auf allen Plattformen und erfreut sich auch beim Raspberry Pi größter Beliebtheit. Außerdem ist Python eine universell einsetzbare Programmiersprache, die sich bei den Aufgaben der Systemadministration ebenso effektiv einsetzen lässt wie im Maschinenbau, der Linguistik, der Pharmaindustrie, in der Banken- und Finanzwelt, der Physik, der Psychologie und vielen anderen Bereichen. Weil bedeutende Firmen wie Google, Facebook, Instagram, Spotify, Quora, Netflix und Dropbox Python benutzen und unterstützen, wird Python auch kontinuierlich weiterentwickelt.

Dieses Buch erscheint nun in der vierten, völlig überarbeiteten Ausgabe. Es eignet sich ebenso für Programmieranfänger als auch für Umsteiger von anderen Programmiersprachen. Behandelt werden nicht nur alle grundlegenden Sprachelemente von Python, sondern auch weiterführende Themen wie Generatoren, Dekorateure, Systemprogrammierung, Threads, Forks, Ausnahmebehandlungen und Modultests. Auf über hundert Seiten wird anschaulich und mit zahlreichen Beispielen auf die vielfältigen Aspekte der Objektorientierung eingegangen.

Brigitte Bauer-Schiewek, Lektorin

[1] Im Vorwort zur 3. Auflage stand hier: Im renommierten PYPL-Index stand Python im Juli 2017 auf dem zweiten Platz hinter Java. Während Java im Vergleich zum Vorjahr jedoch -1,1 % Anteil verloren hatte, gewann Python +4,0 % hinzu.

Danksagung

Zum Schreiben eines Buches benötigt es neben der nötigen Erfahrung und Kompetenz im Fachgebiet vor allem viel Zeit. Zeit außerhalb des üblichen Rahmens. Zeit, die vor allem die Familie mitzutragen hat. Deshalb gilt mein besonderer Dank, wie bereits bei allen vorigen Auflagen auch, meiner Frau Karola, die mich während dieser Zeit tatkräftig unterstützt hat.

Außerdem danke ich den zahlreichen Teilnehmerinnen und Teilnehmern an meinen Python-Kursen, die mir geholfen haben, meine didaktischen und fachlichen Kenntnisse kontinuierlich zu verbessern. Ebenso möchte ich den Besucherinnen und Besuchern meiner Online-Tutorials unter www.python-kurs.eu und www.python-course.eu danken, vor allem jenen, die sich mit konstruktiven Anmerkungen bei mir gemeldet haben. Wertvolle Anregungen erhielt ich auch von denen, die das Buch vorab Korrektur gelesen hatten: Stefan Günther für die Erstauflage des Buches. Für die Hilfe zur zweiten – weitestgehend überarbeiteten – Auflage möchte ich im besonderen Maße Herrn Jan Lendertse und Herrn Vincent Bermel Dank sagen. Für die dritte – wiederum stark veränderte – Auflage danke ich Herrn Wilhelm Wall für seine wertvolle Hilfe, insbesondere Tests unter macOS.

Mein besonderer Dank für die vierte Auflage gilt zum einen Herrn Tobias Habermann und zum anderen Herrn Dr. Konrad Wienands. Herr Habermann hat dafür gesorgt, dass alle Python-Beispiele dieser Auflage automatisch mittels „Pythontex" getestet und ausgeführt werden. Herr Dr. Wienands ist mit großem Sachverstand in die Rolle eines potenziellen Anfängers geschlüpft und hat mir viele wertvolle Hinweise gegeben, wo es möglicherweise offene Fragen oder Probleme bei Neulingen geben könnte. Außerdem hat er auch kleine Unstimmigkeiten gefunden.

Vergessen darf ich auch nicht all diejenigen, die die Exemplare der vorigen Auflagen gekauft und damit erst den Erfolg des Buches ermöglicht hatten. Ebenso danke ich all denjenigen, die mich in E-Mails auf kleine Fehler oder Unstimmigkeiten hingewiesen haben, die ich aber hier leider nicht alle namentlich erwähnen kann.

Zuletzt danke ich auch ganz herzlich dem Hanser Verlag, der dieses Buch – nun auch in der vierten Auflage – ermöglicht. Vor allem danke ich Frau Brigitte Bauer-Schiewek und Frau Kristin Rothe, Programmplanung Computerbuch, für die kontinuierliche ausgezeichnete Unterstützung. Für die technische Unterstützung bei LaTeX-Problemen danke ich Herrn Stephan Korell und Frau Irene Weilhart. Herrn Jürgen Dubau danke ich fürs Lektorat.

Bernd Klein, Singen

Teil I

Einleitung

1 Einleitung

1.1 Einfach und schnell zu lernen

Python ist eine Programmiersprache, die in vielerlei Hinsicht begeistert. Ein riesiger Vorteil im Vergleich zu anderen Sprachen liegt in der leichten Erlernbarkeit der Sprache. In der ersten Auflage führte das Buch den Untertitel „In einer Woche programmieren lernen". Kann man wirklich in so kurzer Zeit programmieren lernen? Das ist möglich! Vor allem mit Python. Wir erfahren es mehrmals im Monat in unseren meist fünftägigen Python-Kursen. Sie werden zum einen von Leuten, die noch keinerlei Programmiererfahrung haben, und zum anderen von Programmierenden mit Erfahrungen in C, C++, Java, Perl und anderen Programmiersprachen besucht. Manche haben auch schon vorher Erfahrungen in Python gesammelt. Aber eines ist in allen Gruppen gleich: Wir haben es immer geschafft, dass jeder programmieren gelernt hat, und vor allen Dingen – was uns am wichtigsten ist – konnten wir immer die Begeisterung für die Sprache Python entfachen.

So ist es auch eines der wichtigsten Ziele dieses Buches, die Leserinnen und Leser möglichst schnell und mit viel Freude zum selbständigen Programmieren zu bringen. Komplexe Sachverhalte werden in einfachen, leicht verständlichen Diagrammen veranschaulicht und an kleinen Beispielen eingeübt, die sich im Laufe vieler Schulungen herausgebildet und bewährt haben.

1.2 Geschichte von Python

Was hat Python mit dem lateinischen Alphabet zu tun? Beide beginnen mit ABC. Im Falle von Python ist es die Programmiersprache ABC. Die Sprache wurde Anfang der 1990er Jahre von Guido van Rossum am Zentrum für Mathematik (Centrum voor Wiskunde en Informatica) in Amsterdam entwickelt. Ursprünglich war Python als Nachfolger für die Lehrsprache ABC entwickelt worden und sollte auf dem verteilten Betriebssystem Amoeba laufen. Guido van Rossum hatte auch an der Entwicklung der Sprache ABC mitgewirkt. In einem Interview mit Bill Venners sagte er: „In den frühen 1980er Jahren arbeitete ich als Implementierer in einem Team, das eine Sprache namens ABC am Centrum voor Wiskunde en Informatica (CWI) aufbaute. Ich weiß nicht, wie gut die Leute den Einfluss von ABC auf Python kennen. Ich versuche, den Einfluss von ABC zu erwähnen, weil ich alles, was

ich während dieses Projekts gelernt habe, und die Leute, die daran gearbeitet haben, zu verdanken habe."[1]

Auch wenn sich das offizielle Logo von Python aus zwei abstrahierten Python-Schlangen zusammensetzt, hat der Name der Programmiersprache Python herkunftsmäßig nichts mit Schlangen zu tun. Für Guido van Rossum stand vielmehr die britische Komikertruppe „Monty Python" mit ihrem legendären „Flying Circus" Pate für den Namen. Guido van Rossum schrieb dazu: „Vor über sechs Jahren, im Dezember 1989, suchte ich nach einem ‚Hobby'-Programmierprojekt, mit dem ich mich während der Weihnachtswoche beschäftigen könnte. Mein Büro (ein staatliches Forschungslabor in Amsterdam) würde geschlossen sein, aber ich hatte einen Computer und nicht viel anderes vor. Ich beschloss, einen Interpreter für die neue Skriptsprache zu schreiben, über die ich in letzter Zeit nachgedacht hatte: einen Nachkommen von ABC, der Unix-/C-Hacker ansprechen würde. Ich habe Python als Arbeitstitel für das Projekt gewählt, da ich in einer etwas respektlosen Stimmung bin (und ein großer Fan von ‚Monty Python's Flying Circus').'[2]

Dennoch sind Assoziationen mit Schlangen möglich und sinnvoll: Man denke nur an das Python-Toolkit „Boa" oder die Programmiersprache Cobra. Außerdem ist eine Schlange, wie schon erwähnt, im Python-Logo.

1.3 Zen von Python

Vom Softwareentwickler Tim Peters stammt das „Zen of Python", welches er 1999 veröffentlichte. Dabei handelt es sich um „Leitprinzipien" zum Schreiben guter Python-Programme. Prinzipien, die sich sowohl an Programmierende als auch an die Designer der Sprache richten und die Sprache Python auch entscheidend prägten:

- Schön ist besser als hässlich.
- Explizit ist besser als implizit.
- Einfach ist besser als komplex.
- Komplex ist besser als kompliziert.
- Flach ist besser als verschachtelt.
- Spärlich ist besser als dicht.
- Lesbarkeit zählt.
- Sonderfälle sind nicht speziell genug, um gegen die Regeln zu verstoßen.
- Obwohl Praktikabilität die Reinheit übertrifft,
- Fehler sollten niemals stillschweigend ablaufen.
- Sofern nicht ausdrücklich zum Schweigen gebracht.
- Widerstehe im Angesicht von Zweideutigkeiten der Versuchung zu raten.

[1] Bill Venners: The Making of Python, A Conversation with Guido van Rossum, Part I, January 13, 2003, https://www.artima.com/intv/python.html

[2] Guido van Rossum schrieb dies in einem Vorwort zur ersten Auflage des Buches „Programming Python" von Marc Lutz.

- Es sollte einen – und vorzugsweise nur einen – offensichtlichen Weg geben, es zu tun.
- Obwohl dieser Weg zunächst vielleicht nicht offensichtlich ist, es sei denn, Sie sind Niederländer.
- Jetzt ist besser als nie.
- Obwohl nie oft besser ist als gerade jetzt.
- Wenn die Implementierung schwer zu erklären ist, ist es eine schlechte Idee.
- Wenn die Implementierung leicht zu erklären ist, kann dies eine gute Idee sein.
- Namespaces sind eine großartige Idee – lasst uns mehr davon machen!

1.4 Zielgruppe des Buches

Beim Schreiben eines Buches stellt man sich ein Gegenüber vor, eine Person, die das Geschriebene lesen, mögen und verstehen soll. Diese Person kann ein beliebiges Geschlecht haben, deswegen haben wir uns im Buch um geschlechtsneutrale Formulierungen bemüht. Begriffe wie „der Nutzer" oder im Plural „die Nutzer" kommen nicht mehr im Buch vor. Allerdings haben wir diese auch nicht durch Formulierungen wie „die NutzerInnen" bzw. „Nutzer*innen" ersetzt, weil diese Formulierungen von einigen abgelehnt werden. Wir denken, dass es uns gelungen ist, geschlechtsneutral zu formulieren, ohne dass der Lesefluss gestört wird oder die Gefühle von manchen verletzt werden.

Natürlich hatten wir beim Schreiben aber nicht nur eine Person im Blickfeld, sondern eine ganze Schar von Lesenden. Da sind zum einen diejenigen, die noch nie programmiert haben und Sachverhalte erklärt haben wollen, die Leute mit Programmiererfahrung in anderen Sprachen vielleicht als „trivial" oder „selbstverständlich" bezeichnen würden. Aber hier ist ein Buch wohl dem Präsenzunterricht, also wenn Lehrende und Lernende am gleichen Ort zusammen sind, deutlich überlegen: Ist einem der Stoff eines Abschnitts oder sogar Kapitels bereits vertraut, kann man es einfach überspringen, bevor man sich zu langweilen beginnt. Ebenso können, falls keine Kenntnisse in anderen Programmiersprachen vorhanden sind, Abschnitte übersprungen werden, in denen wir Ähnlichkeiten, aber auch generelle Unterschiede in der Vorgehensweise von Python im Vergleich zu anderen Sprachen herausarbeiten.

In der obigen Aufzählung fehlt aber noch eine wichtige Gruppe, nämlich diejenigen, die schon Erfahrungen mit Python haben. Dies ist eine Gruppe mit einer breiten Streuung: angefangen bei denjenigen, die bereits ein wenig reingeschnuppert haben, gefolgt von solchen, die bereits kleine oder auch größere Programme geschrieben haben, bis hin zu jenen, die sich als Experten bezeichnen. Unsere Erfahrungen zeigen, dass auch einige aus der zuletzt genannten Gruppe neue Einblicke und Sachverhalte in diesem Buch finden werden, die sich positiv auf deren zukünftige Programmierung mit Python auswirken.

Ansonsten kann dieses Buch auch bestens als Nachschlagewerk benutzt werden. Der umfangreiche Index in diesem Buch macht das Auffinden besonders einfach und erlaubt es damit, dieses Buch außerdem als Referenz zu verwenden, auch wenn es keinesfalls unter diesem Aspekt geschrieben worden ist.

1.5 Aufbau des Buches

Dieses Buch besteht aus sechs Teilen:

Teil I: Wir haben uns entschieden, die Einleitung, in der Sie sich gerade befinden, als eigenen Teil des Buches aufzunehmen. Schließlich enthält sie wichtige Informationen und sollte nach Möglichkeit auch gelesen werden.

Teil II: Im zweiten Teil behandeln wir die Grundlagen der Sprache. Wir lernen Variablen und deren Besonderheit in Python kennen. In diesem Zusammenhang lernen wir auch die Datenstrukturen wie ganze Zahlen (Integers), Fließkommazahlen (Floats), Zeichenketten (Strings), Listen, Tupel, Dictionaries und Mengen kennen. Wir beschäftigen uns eingehend mit allen Anweisungsarten von Python, d.h. Zuweisungen, Kontrollstrukturen wie bedingten Anweisungen und Schleifen sowie Möglichkeiten der Ein- und Ausgabe. Außerdem behandeln wir eingehend Funktionen, Definitionen, einfache und auch komplexere Beispiele, Parameterübergabe sowie Gültigkeitsbereiche von Variablen, und in einem umfangreichen Kapitel gehen wir auch auf rekursive Funktionen ein. Die Modularisierung in Modulen und Paketen ist ein weiterer Themenschwerpunkt.

Teil III: Den dritten Teil des Buches haben wir – entsprechend ihrer Bedeutung – ganz der Objektorientierung gewidmet. In anschaulichen Beispielen führen wir sanft in das objektorientierte Denken und Programmieren ein und zeigen die Besonderheiten von Python deutlich auf. Danach fahren wir mit den fortgeschrittenen Themen der Objektorientierung fort und behandeln neben Vererbung, Mehrfachvererbung und Slots auch extrem fortgeschrittene Themen wie Metaklassen und abstrakte Klassen.

Teil IV: Viele lieben Python, weil es neben der objektorientierten Programmierung auch wesentliche Konzepte der funktionalen Programmierung unterstützt. Deshalb widmen wir diesem Thema den vierten Hauptteil des Buches. Dieser Teil des Buches wendet sich eher an Personen mit Programmiererfahrung, aber es lohnt, sich die behandelten Konzepte anzueignen. Sie sind notwendig, um Python richtig zu verstehen. Die Listen-Abstraktionen bieten ebenso wie die `lambda`-, `map`-, `filter`- und `reduce`-Funktionen eine faszinierende Möglichkeit, die Programmierung auf ein abstrakteres Level zu bringen. Dadurch kann man komplexere Probleme mit geringerem Programmieraufwand lösen, was außerdem zu einer besseren Verständlichkeit und einfacheren Wartbarkeit der Programme führt. Dennoch kann man die gleichen Programme auch ohne diese Techniken schreiben. Eingehend befassen wir uns auch mit dem Themenkomplex „Generatoren und Iteratoren" sowie „Dekoration und Dekorateure"!

Teil V: Im fünften Teil mit dem Titel „Weiterführende Themen " verlassen wir das „eigentliche" Python und wenden uns weiterführenden Themen der Programmierung zu. Diesen Teil hätten wir auch mit „Anwendungsprogrammierung" überschreiben können. Testverfahren und Debugging gehört zu den Fragestellungen, die alle Programmiererinnen und Programmierer bewegen sollten. Im Prinzip hätten wir dieses Kapitel ebenso gut in den ersten Teil nehmen können. Außerdem behandeln wir in einem Kapitel „Reguläre Ausdrücke", die nahezu unerlässlich sind, wenn man Textverarbeitung effizient betreiben will. Man braucht sie beispiels-

weise, wenn man aus Log- oder Parameterdateien bestimmte Informationen herausfiltern will. Den ab Python 3.5 eingeführten Typanmerkungen (*type annotations*) haben wir auch ein eigenes Kapitel gewidmet. In einem weiteren Kapitel zeigen wir außerdem, wie man Daten strukturerhaltend übers Programmende hinaus konservieren kann. Das Kapitel zur Systemprogrammierung dürfte von besonderer Bedeutung für Leute sein, die gerne mit dem Betriebssystem agieren möchten, damit sie ihre Systemprogramme zukünftig unter Python und nicht mehr mit Shell-Skripting verfassen können.

Teil VI: Programmieren lernen ist vor allen Dingen eine aktive Tätigkeit. Nur ein Buch zu lesen und Beispiele nachzuvollziehen, genügt nicht. Deshalb finden Sie zu den meisten Kapiteln interessante und lehrreiche Übungsaufgaben, die den Stoff vertiefen. Aufgaben ohne Lösungen wären aber wenig sinnvoll, weshalb sich im letzten Teil dieses Buches ausführliche Musterlösungen mit Erläuterungen zu den Aufgaben befinden.

Teil VII: Auch wenn es nicht als eigener Teil aufgeführt ist, wollen wir hier noch auf das Stichwortverzeichnis eingehen. Erst ein umfangreiches Stichwortverzeichnis macht aus einem guten Buch ein nützliches und brauchbares Buch!

1.6 Programmieren lernen „interaktiv”

Wie bereits erwähnt, floss in dieses Buch die jahrelange Erfahrung sowohl in der Theorie und Praxis des Programmierens allgemein, aber vor allem auch die Vermittlung des Stoffes in zahlreichen kleinen und großen Kursen mit unterschiedlichsten Besuchertypen ein. Aber ein Buch zu schreiben, stellt dennoch eine neue Herausforderung dar. Beim Buch fehlt leider die direkte Interaktion zwischen dem, der das Wissen vermittelt, und dem Lernenden. Vor Ort kann man sofort sehen, wenn sich bei einem Teilnehmer die Stirn runzelt, große Fragezeichen erscheinen oder wenn sich jemand aus Stress das Ohrläppchen zu reiben beginnt. Dann weiß man als erfahrener Dozent, dass es höchste Zeit ist, ein paar zusätzliche Beispiele und Analogien zu verwenden, oder dass man den Stoff nochmals in anderen – möglicherweise auch einfacheren – Worten erklären sollte. Beim Schreiben eines Buches muss man diese möglichen Klippen vorhersehen und die nötigen zusätzlichen Übungen und Beispiele an den entsprechenden Stellen bereitstellen. Aber was bei vielen Lesern hilft, diese Klippen zu umgehen, führt bei anderen nun vielleicht zu Langeweile und Ungeduld, denn sie empfinden diese zusätzlichen Erklärungen, Übungen oder Beispiele möglicherweise als Zeit- oder Platzvergeudung.

Das Grundproblem ist, dass es sich bei einem Buch nicht um ein interaktives Medium handelt. Aber dank des Internets können wir Ihnen diese Interaktivität dennoch bieten. Im nächsten Abschnitt finden Sie die Adressen, wo Sie Hilfe und zusätzliche Informationen zum Buch finden.

1.7 Download der Beispiele und Hilfe

Alle im Buch verwendeten Beispiele finden Sie zum Download unter

http://www.python-kurs.eu/buch/beispiele/

Auch wenn wir das Buch so geschrieben haben, dass Sie ohne zusätzliche Hilfe auskommen sollten, wird es dennoch hier und da mal ein Problem geben, wo Sie sich vielleicht festgebissen haben. Wenn das Glück es so will, dass Sie in einer Umgebung arbeiten, in der es andere Python-Programmierende gibt, haben Sie es natürlich gut. Aber viele Leserinnen und Leser genießen diesen Vorteil nicht. Dann könnte sich ein Besuch unserer Website besonders lohnen:

http://www.python-kurs.eu/buch/

Dort finden Sie ein Korrekturverzeichnis, zusätzliche bzw. aktualisierte Übungen und sonstige Hilfestellungen. Ansonsten bleibt natürlich immer noch der Einsatz einer Suchmaschine!

1.8 Anregungen und Kritik

Falls Sie glauben, eine Ungenauigkeit oder einen Fehler im Buch gefunden zu haben, können Sie auch gerne eine E-Mail direkt an den Autor schicken: klein@python-kurs.eu.

Natürlich gilt dies auch, wenn Sie Anregungen oder Wünsche zum Buch geben wollen. Leider können wir jedoch – so gerne wir es auch tun würden – keine individuellen Hilfen zu speziellen Problemen geben.

Wir werden versuchen, Fehler und Anmerkungen in kommenden Auflagen zu berücksichtigen. Selbstverständlich aktualisieren wir damit auch unsere Informationen unter

http://www.python-kurs.eu/buch/

Teil II

Grundlagen

2 Kommandos und Programme

2.1 Erste Schritte

2.1.1 Linux

Bei den meisten Linux-Distributionen ist Python bereits vorinstalliert, und damit ist auch die interaktive Python-Shell direkt verfügbar. Der Interpreter wird üblicherweise im Verzeichnis /usr/bin/ (aber manchmal auch in /usr/local/bin) installiert, aber diese Information benötigen wir jetzt noch nicht.

Der einfachste Weg, um mit Python unter Linux zu arbeiten, besteht darin, dass man zuerst ein Terminal startet wie beispielsweise ein „xterm" oder ein „gnome-terminal".

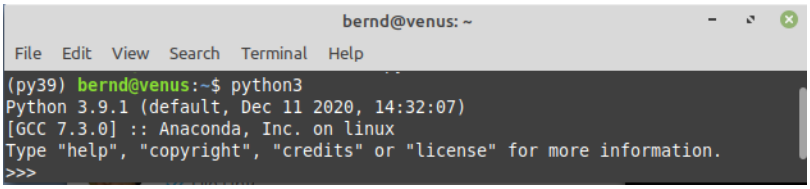


Bild 2.1 Gnome-Terminal

In diesem Terminal – im Bild sehen Sie ein Gnome-Terminal – tippen Sie „python3" ein und drücken die Eingabetaste. Damit starten Sie die interaktive Python-Shell. Python meldet sich mit Informationen über die installierte Version. Der Interpreter steht nun mit dem sogenannten Eingabeprompt (>>>) für Eingaben bereit. Man kann jetzt beliebigen Python-Code eingeben, der nach dem Quittieren mit der Eingabetaste (Return-Taste) sofort ausgeführt wird.

```
(py39) bernd@venus:~$ python3
Python 3.9.1 (default, Dec 11 2020, 14:32:07)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Bei obiger Ausgabe erscheint das Wort „Anaconda", weil auf dem Rechner, auf dem das Kommando ausgeführt worden ist, die Distribution Anaconda installiert ist.[1] Ist kein Anaconda installiert, könnte die Ausgabe beispielsweise wie im Folgenden aussehen:

```
$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2.1.2 Windows

Unter Windows muss Python erst installiert werden. Empfehlenswert ist die Anaconda-Distribution:[2]

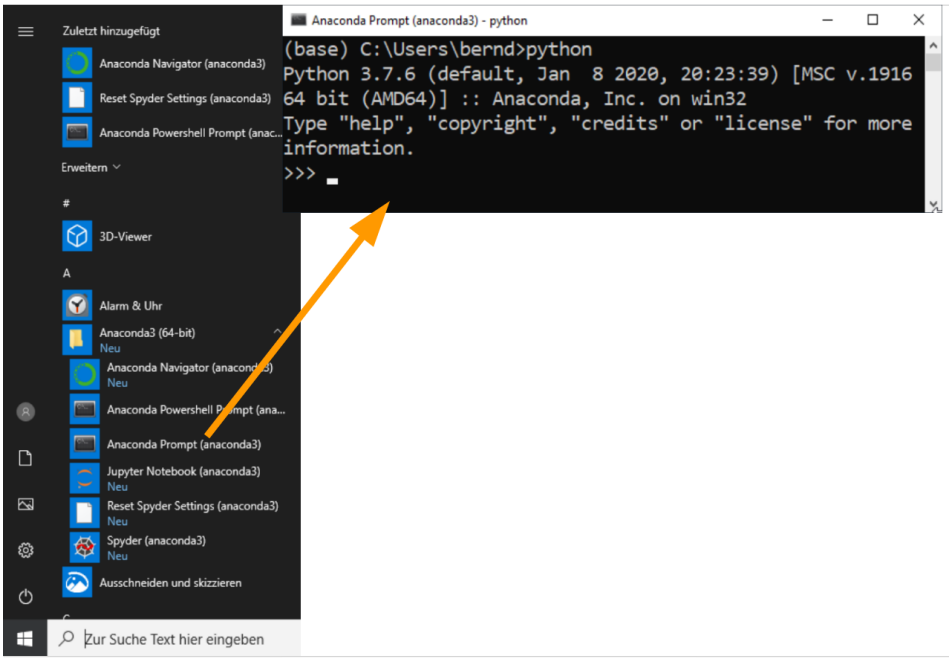


Bild 2.2 Python-Shell unter Windows

Nachdem Python mittels Anaconda installiert worden ist, findet man unter „Alle Programme" eine Rubrik „Anaconda3". Dort kann man dann die „Anaconda Prompt" starten. Nach dem Start müssen wir in dieser den Aufruf `python` starten (siehe Bild 2.2.)

[1] Wir empfehlen, Anaconda zu installieren, was aber für das Buch nicht notwendig ist! Weiteres unter https://www.anaconda.com/

[2] https://www.anaconda.com/distribution/

2.1.3 macOS

Die Installation von Python auf einem Macintosh unter macOS verläuft ähnlich wie bei anderen Unix- und Linux-Plattformen. Die neueren macOS-Versionen werden bereits mit einem vorinstallierten Python ausgeliefert. Allerdings handelt es sich dabei um eine 2.7-Version. Für das Buch benötigen wir jedoch eine 3er-Version von Python. Die neueste Version kann man sich auch von der python.org-Website unter www.python.org/downloads herunterladen.

Nach der Installation von Python3 kann man, wie bei Linux auch, ein Terminal starten und dort mit der Eingabe von `python3` die interaktive Shell starten.[3]

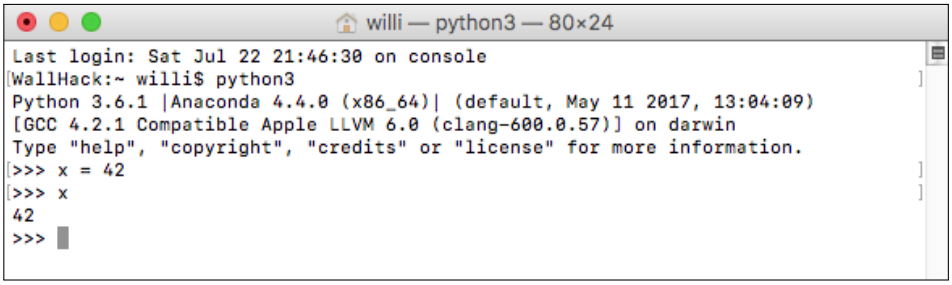


Bild 2.3 Python starten unter macOS

2.2 Interaktive Shell

Wenn die Aktivitäten des vorigen Kapitels erfolgreich waren, sollten Sie sich nun sowohl unter Linux als auch unter Windows in der interaktiven Python-Shell befinden.

Der Begriff „interaktiv" kommt vom lateinischen „inter agere". Das Verb „agere" bedeutet unter anderem „tun", „treiben", „betreiben", „handeln", und „inter" bezeichnet die zeitliche und örtliche Position zu Dingen und Ereignissen, also „zwischen" zwei oder „inmitten" (oder „unter") vielen Objekten, Personen oder Ereignissen sein. Also heißt „inter agere" dazwischen handeln. In diesem Sinne steht die interaktive Shell zwischen dem Anwender und dem Betriebssystem (Linux, Unix, Windows oder anderen) bzw. dem zu interpretierenden Programm (z.B. Python). Die interaktive Shell steht aber auch zeitlich zwischen den einzelnen Aktionen, d.h. zwischen den einzelnen Befehlen und Kommandos. In anderen Worten: Die Shell wartet auf einen Befehl. Sie führt diesen nach der Betätigung der Eingabetaste unverzüglich aus, liefert sofort ein Ergebnis[4] zurück und wartet sofort wieder auf den nächsten Befehl.

Im Englischen steht das Wort *shell* für eine Schale, einen Panzer oder ganz allgemein eine Hülle oder Schutzhülle. *shell* bezeichnet auch ein Schneckenhaus und das Gehäuse, das ei-

[3] Wie man auf dem Bild sieht, haben wir das Anaconda-Paket der Firma Continuum installiert, da es noch über viele zusätzliche Pakete verfügt. Diese Pakete werden allerdings im Rahmen dieses Buches nicht benötigt.

[4] Es könnte auch eine leere Ausgabe sein.

ne Muschel umschließt. Ebenso liegt eine Shell auch zwischen einem Betriebssystem und den Benutzenden. Wie eine Muschelschale schützt sie einerseits das Betriebssystem vor menschlichen Aktionen, und andererseits erspart sie den programmierenden Menschen die Benutzung der „primitiven" und schwer verständlichen Basisfunktionen, indem es ihnen komfortable Befehle zur Kommunikation mit dem Computer zur Verfügung stellt.

Auch die Programmiersprache Python bietet dem Anwender eine komfortable Kommandozeilenschnittstelle, die sogenannte Python-Shell, manchmal auch als interaktive Python-Shell bezeichnet. Man könnte meinen, dass es sich bei dem Begriff „interaktive Shell" um eine Tautologie handelt, da ja, so wie oben beschrieben, Shells immer interaktiv sind. Dies ist jedoch nicht so: Es gibt auch vor allem im Umfeld von Linux und Unix Shells, die als Subshell aufgerufen und nicht interaktiv ausgeführt werden.

2.2.1 Erste Schritte in der interaktiven Shell

Wir nehmen nun an, dass Sie entweder unter Linux oder unter Windows vor einer lauffähigen Python-Shell sitzen, die mit blinkendem Cursor hinter dem Prompt „>>>" auf Ihre Eingabe wartet.

Unsere Experimente mit der Python-Shell starten wir mit der Eingabe eines beliebigen arithmetischen Ausdrucks wie z.B. „4 * 5.2" oder „4 * 12 / 3". Ein Ausdruck wird mit dem Drücken der Eingabetaste (Return) sofort ausgeführt, und das Ergebnis wird in der nächsten Zeile ausgegeben:

```
>>> 4 * 5.3
21.2
>>> 4 * 12 / 3
16.0
```

Die interaktive Shell erlaubt eine Bequemlichkeit, die innerhalb eines Python-Skripts nicht zulässig ist. In einem Programm hätten wir das Ergebnis des obigen Ausdrucks nur mittels einer `print`-Anweisung ausgeben können:

```
>>> print(4 * 5.3)
21.2
```

Python kennt wie die meisten anderen Programmiersprachen die Regel „Punktrechnung geht vor Strichrechnung", wie wir im folgenden Beispiel sehen können:

```
>>> 1 + (42 * 2)
85
>>> 1 + 42 * 2
85

>>> (1 + 42) * 2
86
```

Der jeweils letzte Ausgabewert wird vom Interpreter in einer speziellen Variablen automatisch gespeichert. Der Name der Variable ist einfach ein Unterstrich, also „_". Das Ergebnis der letzten Berechnung kann man sich also damit wieder ausgeben lassen:

```
>>> _
86
```

Der Unterstrich kann im Prinzip wie eine normale Variable benutzt werden:

```
>>> _ * 3
258
```

Dies gilt allerdings nicht in einem Python-Skript oder Python-Programm! In einem Python-Programm hat der Unterstrich nicht diese Bedeutung.

Auch Strings lassen sich mit oder ohne `print` in der interaktiven Shell ausgeben:

```
>>> print("Hello World")
Hello World
>>> "Hello World"
'Hello World'
```

Die Python-Shell bietet noch einen Komfort, den man von den Linux-Shells gewöhnt ist. Es gibt einen Befehlsstack, in dem alle Befehle der aktuellen Sitzung gespeichert werden. Mit den Tasten „↑" und „↓" kann man sich in diesem Stack nach oben („↑"), also zu älteren Befehlen, und nach unten („↓"), also wieder in Richtung neuester Befehl, bewegen.

2.3 Verlassen der Python-Shell

Wir haben zwar gerade erst damit begonnen, in der Python-Shell „herumzuspielen", dennoch wollen möglicherweise einige das Programm bereits wieder verlassen. Es könnte gut sein, dass diesen als Erstes die Tastenkombination Strg-C bzw. Ctrl-C[5] einfällt, aber sie werden schnell merken, dass dies nicht den gewünschten Effekt zeigt. Den Python-Interpreter kann man mit der Kontrollsequenz Strg-D bzw. Ctrl-D wieder verlassen. Alternativ dazu kann man die Python-Shell mittels Eingabe der Funktion `exit()` verlassen.

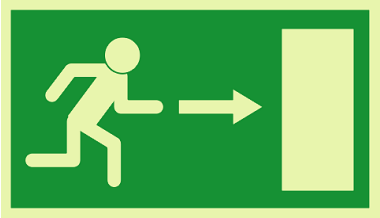


Bild 2.4 Emergency Exit

Alle Einstellungen, wie beispielsweise Variablen oder eingegebene Funktionsdefinitionen, gehen beim Verlassen der Python-Shell verloren und sind beim nächsten Start nicht mehr vorhanden.

2.4 Benutzung von Variablen

In der Python-Shell kann man auch ganz einfach Variablen benutzen. Wenn Sie genau wissen wollen, was es mit Variablen und Datentypen auf sich hat, empfehlen wir, in Kapitel 4 (Datentypen und Variablen) weiterzulesen.

[5] Auf englischen Tastaturen steht Ctrl statt Strg.

Das folgende Beispiel wendet sich an diejenigen, die bereits ein wenig Erfahrung mit Programmierung haben. Man kann also ganz einfach Werte in Variablen speichern. Variablennamen bedürfen keiner besonderen Kennzeichnung wie in anderen Sprachen wie zum Beispiel das Dollarzeichen in Perl:

```
>>> maximal = 124
>>> breite = 94
>>> print(maximal - breite)
30
```

Die Anweisung `maximal = 124` sollte man als „die Variable `maximal` soll den Wert `124` referenzieren" lesen bzw. interpretieren.

Alle Variablennamen können aus einer Folge von beliebigen Buchstaben und Ziffern sowie dem Unterstrich („_") bestehen. Sie dürfen allerdings nicht mit einer Ziffer beginnen. Python unterscheidet zwischen Groß- und Kleinschreibung, d.h. eine Variable mit dem Namen `breite` ist eine andere Variable als die mit dem Namen `Breite`, wie wir im Folgenden sehen:

```
>>> breite = 94
>>> Breite = 42
>>> print(breite)
94
>>> print(Breite)
42
```

2.5 Mehrzeilige Anweisungen

Bis jetzt haben wir noch nicht über mehrzeilige Anweisungen gesprochen. Anfänger werden hier vielleicht Probleme haben, da noch einige Grundlagen zum Verständnis fehlen, und sollten deshalb besser mit dem folgenden Kapitel weitermachen.

Wir demonstrieren, wie die interaktive Shell mit mehrzeiligen Anweisungen wie zum Beispiel for-Schleifen umgeht.

```
>>> l = ["A", 42, 78, "Just a String"]
>>> for character in l:
...     print(character)
...
A
42
78
Just a String
```

Achtung: Nachdem man die Zeile `for character in l:` eingetippt hat, wartet die interaktive Shell im Folgenden auf eingerückte Zeilen. Dies teilt uns die Shell mit einem neuen Prompt mit: Statt dem gewohnten „>>>" erhalten wir nun eine Folge von drei Punkten „...". Wir müssen also alle Anweisungen, die mit der Schleife ausgeführt werden sollen, um die gleiche Anzahl von Leerzeichen einrücken, mindestens jedoch ein Leerzeichen!

Wenn wir fertig sind, müssen wir nach der letzten Code-Zeile zweimal die Eingabetaste drücken, bevor die Schleife wirklich ausgeführt wird.[6]

2.6 Programme schreiben

Wir haben bisher ein wenig mit der interaktiven Python-Shell herumprobiert, aber noch kein richtiges Programm geschrieben. Bei der Programmierung ist es generell wichtig, möglichst in kleinen Schritten vorzugehen. Deshalb wird unser erstes Programm in Python auch nur sehr wenig tun. Das Programm soll lediglich einen kurzen Text ausgeben, nachdem es gestartet worden ist. Für diesen Text wird in fast allen Einführungen zu Programmiersprachen die Zeichenkette[7] „Hallo Welt” oder auf Englisch „Hello World” verwendet. Wir wollen dieser Tradition folgen.

Bild 2.5 Hallo Welt

Im Folgenden wollen wir zeigen, was in der Programmiersprache C++ nötig ist, um ein so kleines Programm zum Laufen zu bringen. Sie können diesen kurzen Exkurs in die Programmiersprache C++ bedenkenlos überspringen. Wir bringen dieses Beispiel nur, damit man später erkennen kann, wie leicht es ist, in Python im Vergleich zu C++ zu programmieren.

Folgender Programmcode ist in C++ nötig, um ein Programm zu schreiben, was nach dem Start nur den Text „Hello World” in einer Zeile ausgibt:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World!\n";
    return 0;
}
```

Um das Programm zu compilieren[8], muss man es in einer Datei speichern, also z.B. „hello.cpp”. Danach kann man mit folgendem Befehl ein lauffähiges Programm unter dem Namen „hello” erzeugen:

6 Aktiv selbst einrücken muss man jedoch nur bei der Python-Standardshell. Benutzt man die „ipython”- oder die „idle”-Shell, erfolgt die Einrückung automatisch!

7 Eine Folge von Zeichen wird als „Zeichenkette” oder meistens als „String” bezeichnet.

8 Näheres zu Compiler, Interpreter, Maschinen und Skriptsprachen erfahren Sie in Kapitel 3 (Bytecode und Maschinencode).

```
$ g++ -o hello hello.cpp
```

Im Folgenden starten wir das Programm „hello” und erhalten die gewünschte Ausgabe:

```
$ ./hello
Hello World!
$
```

Deutlich einfacher können wir ein solches Programm unter Python erstellen.

Schreiben Sie in einem Editor[9] Ihrer Wahl die folgende Zeile und speichern Sie sie unter dem Namen „hello.py” ab:

```
print("Hello World")
```

Unter Linux oder Unix starten wir dieses Programm am einfachsten, wenn wir uns in einem Terminal in das Verzeichnis begeben, in dem wir die Datei hello.py abgespeichert haben. Dort können wir dann das Programm mit Kommando `python3 hello.py` ausführen. Im Folgenden sehen wir dieses Vorgehen inklusive Ergebnis. Dabei nehmen wir an, dass Sie die Datei in einem Unterverzeichnis `python_beispiele` Ihres Home-Verzeichnisses abgespeichert hatten:

```
$ cd ~/python_beispiele/
$ python3 hello.py
Hallo Welt
```

Unter Windows können wir ähnlich vorgehen. Wir starten erst das Programm „Eingabeaufforderung”. Dort bewegen wir uns in unser Verzeichnis mit den Python-Programmen, in das wir unser hello.py abgespeichert haben.

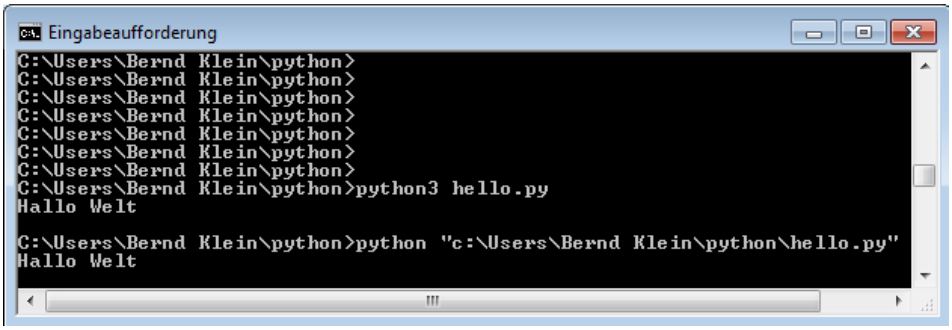


Bild 2.6 Hallo Welt unter Python in der Windows-Eingabeaufforderung

[9] Unter Windows könnte das beispielsweise „Notepad” sein, unter Linux oder einem anderem Unix-System empfiehlt sich der vi, emacs, kate oder gedit.

Wir haben gesehen, dass die nötige Syntax[10] unter Python deutlich geringer ist als unter C++, d.h. man muss lediglich den Text mit einer print-Funktion aufrufen. Anschließend ist auch kein explizites Compilieren[11] nötig. Man kann das Programm einfach so starten.

Fragen zum Verständnis:

- Wie kann man ein Python-Programm schreiben?
- Wie startet man ein Python-Programm?
- Wie kann man das Produkt aus 3.5 und 7.8 am einfachsten mit Python bezeichnen?

[10] Etymologisch leitet sich das Wort Syntax aus dem altgriechischen Wort „syntaksis” ab. „syn” bedeutet „zusammen”, und „taksis” steht für „Reihenfolge” oder „Ordnung”. Unter Syntax versteht man in der Linguistik die Lehre vom Satzbau, also sozusagen die „zusammen”- oder „gesamt”-Reihenfolge der einzelnen Satzteile. In der Informatik versteht man unter der Syntax einer Programmiersprache die Regeln oder das System von Regeln, mit denen man gültige Programme der Sprache beschreiben kann.

[11] Man beachte das Wort „explizit”. Beim Start des Programmes wird nämlich implizit gegebenenfalls eine Compilierung vorgenommen. Auf diese gehen wir im nächsten Kapitel näher ein.

3 Bytecode und Maschinencode

3.1 Einführung

Dieses Kapitel können Sie beim Einstieg in die Materie auch gerne überspringen, da das Verständnis für die folgenden Kapitel nicht notwendig ist. Diejenigen, die noch nie programmiert haben, sollten dieses Kapitel definitiv überspringen. Hier geht es um allgemeine Fragen: Handelt es sich bei Python um eine Programmier- oder eine Skriptsprache? Wird ein Python-Programm übersetzt oder interpretiert? Kann der gleiche Python-Code auf verschiedenen Rechnern oder Betriebssystemen laufen? Worin liegt der Zusammenhang zwischen Python, Jython oder Java?

3.2 Unterschied zwischen Programmier- und Skriptsprachen

Die Frage, worin der Unterschied zwischen Programmier- und Skriptsprachen liegt, lässt sich nicht einfach beantworten. Sehr häufig werden die Begriffe Skript und Programm nahezu synonym verwendet. Es gibt Fälle, in denen es eindeutig ist: Hat man beispielsweise ein Problem in C gelöst, so wird allgemein das Ergebnis als „Programm" bezeichnet. Kaum jemand würde ein C-Programm als Skript bezeichnen. Bei Shells wie z.B. der Bourne- oder der Bash-Shell spricht man nahezu immer von Skripten und nicht von Programmen. Würde man von einem Bourne-Shell-Programm sprechen, so klingt das mindestens so falsch, als bezeichnete man ein Fahrrad als Moped. Worin der Unterschied zwischen einem Shell-Skript und einem Programm liegt, wird in Tabelle 3.1 auf der nächsten Seite gezeigt.

3.3 Interpreter- oder Compilersprache

Unter einer Compilersprache versteht man eine Programmiersprache, deren Programme vor ihrer Ausführung vollständig in Maschinencode, also Binärcode, übersetzt werden. Dies kann leicht zu Verwechslungen führen, da manchmal auch die Sprache, in der ein

Tabelle 3.1 Unterschiede zwischen Skripten und Programmen

Skripte	Programme
Ein Skript besteht typischerweise nur aus „wenigen" Zeilen Code, häufig weniger als 100 Zeilen.	Programme können sehr klein sein, also beispielsweise weniger als Hundert Zeilen Code, aber sie können auch aus mehreren Tausend oder gar Millionen Codezeilen bestehen.
Skripte werden unmittelbar ausgeführt. Sie werden nicht übersetzt (bzw. compiliert), sondern interpretiert.	Programme in Programmiersprachen wie C und C++ müssen immer in Maschinencode übersetzt werden, bevor sie ausgeführt werden können.

Compiler geschrieben wird, als Compilersprache bezeichnet wird. Nehmen wir an, dass wir eine Programmiersprache XY[1] hätten, für die es einen Compiler cXY gibt. Dann bezeichnet man XY als eine Compilersprache, weil sie mittels cXY in Maschinencode übersetzt wird. Wurde nun der Compiler[2] cXY in der Programmiersprache C geschrieben, dann sagt man „C ist die Compilersprache von XY".

Bei einer Interpretersprache werden die Programme Befehl für Befehl interpretiert und ausgeführt.

Die Programmiersprachen C und C++ sind reine Compilersprachen, d.h. jedes Programm muss zuerst in Maschinencode übersetzt werden, bevor es ausgeführt werden kann. Sprachen wie Basic und VB werden interpretiert.

Man liest leider allzu häufig, dass es sich bei Python um eine interpretierte Sprache handelt. Dies ist nicht richtig. Aber bei Python handelt es sich auch nicht um eine Compilersprache im „klassischen" Sinne. Python-Programme werden sowohl übersetzt als auch interpretiert. Startet man ein Python-Programm, wird zuerst der Programmcode in Bytecode übersetzt. Bei Bytecode handelt es sich um Befehle für eine virtuelle Maschine. Bei Python spricht man von der PVM (Python Virtual Machine).[3] Dieser Code ist unabhängig von realer Hardware, aber kommt dem Code von Maschinensprachen schon relativ nahe. Dieser Bytecode wird dann für einen bestimmten Maschinencode interpretiert. Das bedeutet, dass der Bytecode plattformunabhängig ist.

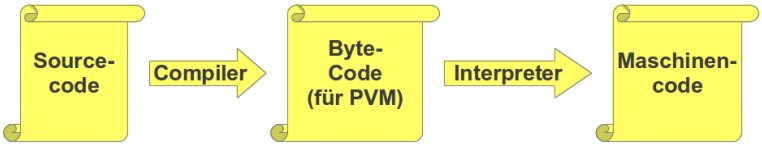


Bild 3.1 Python: Compiler und Interpreter

Importiert man ein Python-Modul, so wird dieses nicht nur für diesen Lauf kompiliert, sondern auch dauerhaft gespeichert. Dadurch kann Python in einem späteren Lauf auf bereits compilierten Code zurückgreifen. Dies natürlich nur, falls sich der Quellcode des Moduls

1 Nach unserem Kenntnisstand gibt es keine Programmiersprache mit diesem Namen.
2 Beim Compiler handelt es sich auch um ein Programm.
3 Dieses Konzept entspricht der Vorgehensweise bei Java. Auch dort wird der Code zuerst in Bytecode übersetzt, d.h. Code für die JVM (Java Virtual Machine).

nicht verändert hat. Nun stellt sich die Frage, wo – also in welchem Verzeichnis – Python den Bytecode speichert. Sie werden in dem Verzeichnis, in dem sich Ihr Modul befindet, ein Verzeichnis `__pycache__` finden. Innerhalb dieses Programms finden Sie dann die compilierten Module, die Sie an der Endung `.pyc` erkennen können.[4]

Es ist einfach, Obiges selbst zu testen. Dazu erzeugen wir eine Datei „beispiel.py" mit folgender Zeile als Inhalt:

```
print("Irgendetwas muss ich halt sagen!")
```

Geben Sie dann im Python-Interpreter folgende Zeile ein:

```
>>> import beispiel
Irgendetwas muss ich halt sagen!
```

In dem Unterverzeichnis `__pycache__` befindet sich nun eine Datei `beispiel.cpython-38.pyc`.

[4] In den Python 2.x-Versionen wurden die compilierten Versionen im gleichen Verzeichnis wie die Module abgespeichert!

4 Datentypen und Variablen

4.1 Einführung

Sie glauben, weil Sie bereits in Sprachen wie C und C++ programmiert haben, dass Sie bereits genug über Datentypen und Variablen wissen? Vorsicht ist geboten! Sie wissen sicherlich viel, aber wahrscheinlich nicht genug, wenn es um Python geht. Deshalb lohnt es sich auf jeden Fall, hier weiterzulesen. Es gibt gravierende Unterschiede zwischen Python und anderen Programmiersprachen in der Art, wie Variablen behandelt werden. Vertraute Datentypen wie Ganzzahlen (Integer), Fließkommazahlen (floating point numbers) und Strings (Zeichenketten) sind zwar in Python vorhanden, aber auch hier gibt es wesentliche Unterschiede zu C/C++ und Java. Dieses Kapitel ist also sowohl für Anfänger als auch für fortgeschrittene Programmierende zu empfehlen.

Eine Variable ist ein Name bzw. ein Bezeichner, mit dem man auf ein Objekt zugreifen kann.[1] Ein Objekt steht zwar an einem bestimmten Speicherplatz, aber in Python spricht und denkt man nicht in physikalischen Speicherplätzen. Eine Variable ist nicht einem festen Objekt oder Speicherplatz zugeordnet. Während des Programmablaufs können einem Variablennamen neue beliebige Objekte zugewiesen werden, d.h. die Variable referenziert nach jeder Zuweisung in den meisten Fällen einen neuen Speicherort. Diese Objekte können beliebige Datentypen sein. Eine Variable referenziert also ein Objekt in Python.

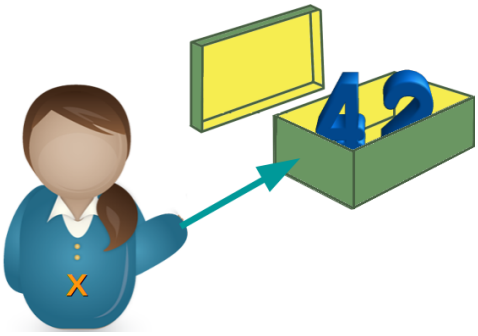


Bild 4.1 Variablen sind Referenzen auf Objekte.

[1] In den meisten Programmiersprachen ist es so, dass eine Variable einen festen Speicherplatz bezeichnet, in dem Werte eines bestimmten Datentyps abgelegt werden können. Während des Programmlaufs kann sich der Wert der Variable ändern, aber die Wertänderungen müssen vom gleichen Typ sein. Also kann man nicht in einer Variable zu einem bestimmten Zeitpunkt eine Integer-Zahl gespeichert haben und dann diesen Wert durch eine Fließkommazahl überschreiben. Ebenso ist der Speicherort der Variablen während des gesamten Laufs konstant, kann also nicht mehr geändert werden.

Im Folgenden kreieren wir eine Variable x in der interaktiven Python-Shell, indem wir ihr einfach den Wert 42 unter Verwendung eines Gleichheitszeichens zuweisen:

```
>>> x = 42
```

Man bezeichnet dies als Zuweisung oder auch als Definition einer Variablen. Genau genommen müsste man sagen, dass wir ein Integer-Objekt „42" erzeugen und es mit dem Namen x referenzieren.

Anmerkung: Obige Anweisung darf man nicht als mathematisches Gleichheitszeichen sehen, sondern als „der Variablen x wird der Wert 42 zugewiesen", d.h. der Inhalt von x ist nach der Zuweisung 42. Man kann die Anweisung also wie folgt „lesen": „x soll sein 42".

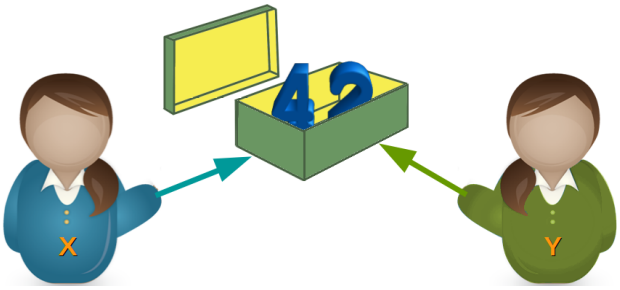
Wir können nun diese Variable zum Beispiel benutzen, indem wir ihren Wert mit der Funktion `print` ausgeben lassen:

```
>>> print("Wert von x: ", x)
Wert von x:  42
```

Wir können sie aber auch auf der rechten Seite einer Zuweisung verwenden:

```
>>> y = x
```

Beide Variablen zeigen nun auf das gleiche Objekt, d.h. das `int`-Objekt 42:



Woher wissen wir oder noch besser, wie kann man beweisen, dass beide Variablen auf das gleiche Objekt zeigen? Dazu bietet sich die `id`-Funktion an. Dies ist eine Funktion, die für ein Objekt die Identität eines Objektes zurückgibt. Erhält man für zwei Variablennamen die gleiche Identität, so weiß man, dass sie das gleiche Objekt referenzieren. Wir benutzen nun `id`, um die obige Behauptung zu beweisen:

```
>>> x = 42
>>> id(x)
9786176
>>> y = x
>>> id(y)
9786176
```

Der Ergebniswert von `id` interessiert in diesem Fall nicht. Wir möchten nur wissen, ob wir für beide Variablen die gleichen Werte erhalten:

```
>>> x = 42
>>> y = x
>>> id(x) == id(y)
True
```

Mit dem Ausdruck `id(x) == id(y)` können wir also prüfen, ob die beiden Variablen das gleiche Objekt referenzieren. Für einen solchen Test bietet Python auch den Operator `is`. Liefert `is` True, handelt es sich um das gleiche Objekt:

```
>>> pi1 = 3.141592653589793
>>> pi2 = 3.141592653589793
>>> pi1 is pi2  #  the same as id(pi1) == id(pi2)
False

>>> pi1 = 3.141592653589793
>>> pi2 = pi1
>>> pi1 is pi2
True
```

Den Operator `is` darf man keinesfalls mit dem Gleichheitsoperator == verwechseln. Mit == kann man überprüfen, ob Objekte gleich sind, was aber nicht impliziert, dass die Objekte identisch sind.[2] Ein Beispiel aus dem täglichen Leben kann dies illustrieren. Wenn jemand sagt: „Das selbe Auto ist schon wieder vorbeigefahren.", dann meint diese Person, dass es sich physikalisch um das gleiche Auto handelt, also gleiches Nummernschild beispielsweise. Wenn jemand sagt „Das ist das gleiche Auto, wie ich es habe!", dann meint die Person nur, dass sie das gleiche Modell besitzt. Im folgenden Code sehen wir, dass s1 und s2 sowohl gleich sind als auch das identische Objekt referenzieren:

```
>>> s1 = "Der Unterschied zwischen 'das Gleiche' und 'dasselbe'"
>>> s2 = s1
>>> s1 == s2
True
>>> s1 is s2
True
```

Nun sehen wir, dass Gleichheit nicht immer Identität impliziert:

```
>>> s2 = "Der Unterschied zwischen 'das Gleiche' und 'dasselbe'"
>>> s1 is s2
False
>>> s1 == s2
True
```

Bisher stand meist nur eine einzelne Variable rechts vom Gleichheitszeichen. Auf der rechten Seite einer Zuweisung kann aber auch ein beliebiger, zu berechnender Ausdruck stehen, dessen Ergebnis dann der Variablen auf der linken Seite zugewiesen wird. Ein solcher Ausdruck kann auch andere Variablennamen enthalten. Diesen muss jedoch zuvor ein Wert zugewiesen worden sein. Das bedeutet, dass anschließend die Variable y auf das Ergebnis der Addition des alten y-Wertes und 36 zeigt.

```
>>> a = 3
>>> b = 1
>>> c = a + b
```

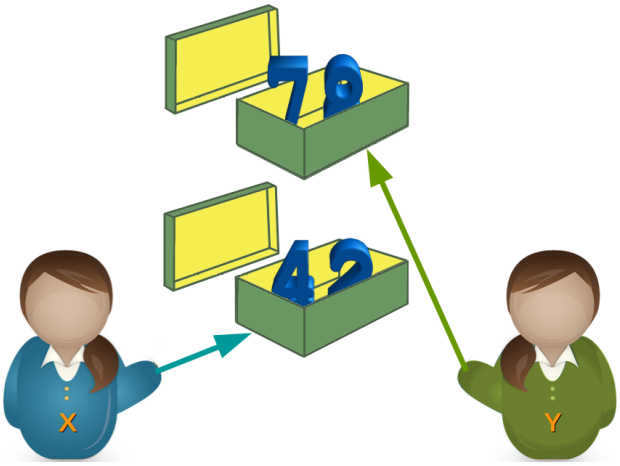
[2] In Gedanken kann man in den folgenden Beispielen auch den Begriff „Objekt" durch „Wert" ersetzen. Wir wollten von Anfang an die korrekten Begriffe, also „Objekt", verwenden.

```
>>> c
4
```

Wie man im nächsten Beispiel sehen kann, ist es möglich, eine Variable gleichzeitig auf der linken und rechten Seite einer Zuweisung zu benutzen.

```
>>> x = 42
>>> y = x
>>> y = y + 36
>>> y
78
```

Für die Schreibweise „`y = y + 36`" verwendet man üblicherweise die sogenannte erweiterte Zuweisung „`y += 36`".



Vom Ergebnis her gesehen sind die beiden Möglichkeiten identisch. Allerdings gibt es einen Implementierungsunterschied. Bei der erweiterten Zuweisung muss die Referenz der Variablen y nur einmal ausgewertet werden.[3]

Erweiterte Zuweisungen gibt es auch für die meisten anderen Operatoren wie „-", „*", „/", „**", „//" (ganzzahlige Division) und „%" (Modulo).

4.2 Variablennamen

Wenn wir uns beim Programmieren Variablennamen ausdenken, muss man zweierlei beachten: Zum einen muss der Name den syntaktischen Anforderungen der Programmiersprache genügen, und zum anderen muss er den üblichen Gepflogenheiten und Konventionen entsprechen.

4.2.1 Gültige Variablennamen

Variablennamen müssen mit einem Buchstaben oder Unterstrich „_" beginnen. Die folgenden Zeichen dürfen sich aus einer beliebigen Folge von Buchstaben, Ziffern und dem

[3] Bei Datentypen wie Listen kann es zu extremen Laufzeitproblemen kommen, wenn man keine erweiterte Zuweisung verwendet. Darauf gehen wir jedoch erst später ein.

Unterstrich zusammensetzen. Variablennamen sind case sensitive.[4] Dies bedeutet, dass Python zwischen Groß- und Kleinschreibung unterscheidet:

```
>>> person_42 = "Marvin"
>>> Person_42 = "Arthur"
>>> print(person_42)
Marvin
>>> print(Person_42)
Arthur
```

Zu den gültigen Buchstaben gehören aber nicht nur „lateinische" Buchstaben, sondern auch alle anderen Unicode-Buchstaben, wie beispielsweise kyrillische, griechische und so weiter.

4.2.2 Konventionen für Variablennamen

Man bevorzugt in der Python-Community Kleinschreibung bei Variablennamen, z.B. `minimum` statt `Minimum`. Außerdem werden Variablennamen, die aus mehreren Wörtern bestehen, mittels Unterstrich (_) separiert, also beispielsweise `minimale_breite`. Variablennamen mit Binnenversalien – bei Programmierenden besser als „camel case" oder „CamelCase" bekannt – sind in der Python-Welt nicht beliebt: `MinimaleBreite` oder `minimaleBreite`.

Ansonsten zeichnet sich ein guter Programmierstil dadurch aus, dass man möglichst sprechende Variablennamen verwendet. Also beispielsweise `aktueller_kontostand` oder `maximale_beschleunigung` statt nichtssagender Namen wie `ak` oder `mb`.

4.3 Datentypen

4.3.1 Ganze Zahlen

Ganze Zahlen werden in der Informatik üblicherweise als Integer bezeichnet. Es handelt sich dabei um die Zahlen ... -3, -2, -1, 0, 1, 2, 3, ...

Während Integer-Zahlen in den meisten Programmiersprachen durch einen Maximalwert begrenzt sind, sind sie in Python unbegrenzt, d.h. sie können beliebig groß bzw. beliebig klein werden. Die Variable `unsigned_long_max` bezeichnet in der folgenden interaktiven Python-Session den maximalen Integer-Wert in C für „unsigned long". Ein Wert, der in C für diesen Datentyp nicht überschritten werden kann. Wir sehen, dass wir in Python diesen Wert sogar ohne Fehler beispielsweise quadrieren können.

```
>>> unsigned_long_max = 4294967295
>>> unsigned_long_max * unsigned_long_max
18446744065119617025
>>> x = 78764738298878908787878236666565434233414300890910006 54
```

[4] Für diesen Begriff gibt es keine deutsche Übersetzung.

```
>>> x * x
620388399929088197813485589212087821899160651152524666622241420288 30
↪ 910197615930689481485927796837531028427716
```

Bei Integer-Zahlen muss man beachten, dass sie nicht mit einer 0 beginnen dürfen, wenn es sich um eine Zahl im Dezimalsystem handeln soll:[5]

```
>>> 42
42
>>> 042
  File "<stdin>", line 1
    042
      ^
SyntaxError: leading zeros in decimal integer literals are not
↪ permitted; use an 0o prefix for octal integers
```

Die führende Null wird benötigt, wenn man Binär-, Oktal- oder Hexadezimalzahlen schreiben möchte. Literale für Binärzahlen beginnen mit der Sequenz `0b` oder `0B` und werden dann von der eigentlichen Binärzahl gefolgt. Die Binärzahl `101010` schreibt man also wie folgt:

```
>>> x = 0b101010
>>> x
42
```

Man sieht, dass dies keine Auswirkung auf die Interndarstellung der Zahl hat.

Literale für Oktalzahlen beginnen mit der Sequenz `0o` oder `0O` und werden dann von der eigentlichen Oktalzahl gefolgt:

```
>>> x = 0o10
>>> x
8
```

Literale für Hexadezimalzahlen beginnen mit der Sequenz `0x` oder `0X` und werden dann von der eigentlichen Hexzahl gefolgt:

```
>>> x = 0x10
>>> x
16
>>> x = 0x1A
>>> x
26
```

Mit den Funktionen `hex`, `bin`, `oct` kann man eine Integer-Zahl in einen String wandeln, der der Stringdarstellung der Zahl in der entsprechenden Basis entspricht:[6]

```
>>> x = hex(19)
>>> x
'0x13'
>>> type(x)
```

[5] Die Fehlermeldung im folgenden Beispiel gilt ab Python 3.8. Vorher wurde der Fehler "SyntaxError: invalid token" erhoben.

[6] Strings, also Zeichenketten, haben wir noch nicht eingeführt. Diese Umwandlung haben wir hier nur der Vollständigkeit wegen eingeführt!

```
<class 'str'>
>>> bin(65)
'0b1000001'
>>> oct(65)
'0o101'
>>> oct(0b101101)
'0o55'
```

4.3.2 Fließkommazahlen

Fließkommazahlen[7] entsprechen dem Datentyp `float` in Python. Es handelt sich dabei um Zahlen der Art 2.34, 27.87878 oder auch 3.15e2:

```
>>> x = 2.34
>>> y = 3.14e2
>>> y
314.0
```

Achtung: In deutschsprachigen Ländern werden die Nachkommastellen im täglichen Leben mit einem Komma eingeleitet. So schreibt man beispielsweise 1,99, wenn ein Artikel 1,99 € kostet. In Python und in anderen Programmiersprachen wird statt des Kommas immer ein Punkt, der sogenannte Dezimalpunkt, benutzt! Wie man aus obigem Beispiel ersehen kann, entspricht die Zahl „3.14e2" dem Ausdruck $3.14 \cdot 10^2$.

4.3.3 Zeichenketten

Wir werden im nachfolgenden Kapitel weiter auf Strings eingehen.

4.3.4 Boolesche Werte

Bei den booleschen Werten handelt es sich um einen Datentyp, der nur die zwei Werte True[8] (wahr) oder False (falsch) annehmen kann. Allerdings entspricht das im Prinzip den numerischen Werten 1 für True und 0 für False. Damit lässt sich mit booleschen Werten auch „numerisch" rechnen, auch wenn das nicht immer gutem Programmierstil entspricht:

```
>>> x = True
>>> x * 4
4
```

Für zwei boolesche Variablen x und y gilt Folgendes:

[7] Sie werden manchmal auch als Gleitpunktzahlen bezeichnet.
[8] Ein beliebter Anfangsfehler besteht darin, dass man nicht auf die Großschreibung der beiden Werte achtet!

Tabelle 4.1 Logisches UND und ODER

Operator	Erklärung
not y	Negierung von y
x and y	Logisches UND von x und y
x or y	Logisches ODER von x und y

Beispielanwendungen:

```
>>> x = True
>>> not x
False
>>> y = False
>>> x and y
False
>>> x or y
True
>>> x and not y
True
```

4.3.5 Komplexe Zahlen

Python bietet einen Datentyp `complex` für komplexe Zahlen, den man bei den meisten Programmiersprachen vergeblich sucht. Allerdings würden ihn die meisten Anwender von Python wohl auch nicht vermissen. Wenn Sie komplexe Zahlen nicht kennen oder nichts mit ihnen zu tun haben, können Sie das Folgende gerne überspringen. Mathematisch gesehen erweitern die komplexen Zahlen die reellen Zahlen derart, dass die Gleichung $x^2 + 1 = 0$ lösbar wird. In der Mathematik werden komplexe Zahlen meist in der Form $a + b \cdot i$ dargestellt, wobei a und b reelle Zahlen sind und i die imaginäre Einheit ist. In Python benutzt man, der Konvention in der Elektrotechnik folgend, ein „j" als imaginäre Einheit.

```
>>> x = 3 + 4j
>>> y = 2 - 4.5j
>>> x + y
(5-0.5j)
>>> x * y
(24-5.5j)
```

4.3.6 Operatoren

Eine Übersicht über die gängigsten Operatoren bietet die Tabelle 4.2. Zwei von diesen Operatoren werden wir nun im Detail vorstellen, nämlich „//" und „%":

„//" bezeichnet die ganzzahlige Division, d.h. `11 // 4` ergibt nicht den exakten Float-Wert `2.75` , sondern den Integer-Wert `2`, d.h. es wird immer auf die nächste ganzzahlige Inte-

Tabelle 4.2 Erklärung zu Operatoren

Operator	Erklärung
x + y	Summe von x und y
x - y	Differenz von x und y
x * y	Produkt von x und y
x / y	Quotient von x und y
x // y	Ganzzahlige Division
x % y	Modulo- oder Restdivision
abs(x)	Betrag von x
x ** y	Potenzieren, also x hoch y

ger „abgerundet". Dies gilt allerdings nur für den Fall, dass beide Operanden ganze Zahlen (`int`) sind! Ist mindestens einer der Operanden eine float-Zahl, wird auch auf die nächste ganzzahlige Integer-Zahl abgerundet, aber dann das Ergebnis in `float` gewandelt:

```
>>> 8 // 3
2
>>> 11 // 3
3
>>> 12 // 3
4
>>> 12.0 // 3
4.0
```

Mit `%` lässt sich der Rest bei der Integerdivison bestimmen: So ergibt `11 % 4` den Wert `3` und `10 % 4` den Wert `2`.

```
>>> 8 % 3
2
>>> 9 % 3
0
>>> 8.0 % 3
2.0
```

Es gilt folgender Zusammenhang zwischen der ganzzahligen Division und dem Modulo-Operator:

```
>>> x = 24
>>> y = 7
>>> x == (x // y) * y + (x % y)
True
```

4.4 Statische und dynamische Typdeklaration

Wer Erfahrungen mit C, C++, Java oder ähnlichen Sprachen hat, hat dort gelernt, dass man einer Variablen einen Typ zuordnen muss, bevor man sie verwenden darf. Der Datentyp muss im Programm festgelegt werden – und zwar, bevor die Variable zum ersten Mal benutzt oder definiert wird. Dies sieht dann beispielsweise in einem C-Programm so aus:

```
int i, j;
float x;
x = i / 3.0 + 5.8;
```

Während des Programmlaufs können sich dann die Werte für i, j und x ändern, aber ihre Typen sind für die Dauer des Programmlaufs auf `int` im Falle von i und j und `float`für die Variable x festgelegt. Dies bezeichnet man als „statische Typdeklaration", da bereits der Compiler den Typ festlegt und während des Programmablaufs keine Änderungen des Typs mehr vorgenommen werden können.

In Python sieht dies anders aus, wie wir bereits die ganze Zeit gesehen haben. Zunächst einmal bezeichnen Variablen in Python keinen bestimmten Typ, und deshalb benötigt man in Python keine Typdeklaration. Benötigt man im Programm beispielsweise eine Variable i mit dem Wert 42 und dem Typ Integer, so erreicht man dies einfach mit der Anweisung „`i = 42`".

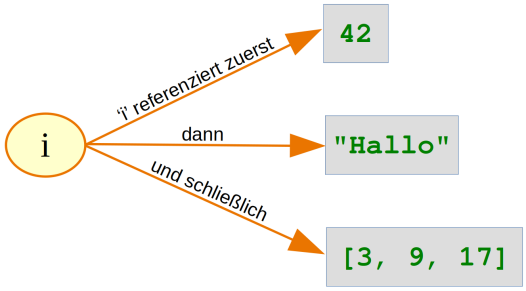


Bild 4.2 ‚i' während der Laufzeit

Damit hat man automatisch ein Objekt vom Typ Integer angelegt, welches dann von der Variablen i referenziert wird. Man kann dann im weiteren Verlauf des Programms der Variablen i auch andere Objekte mit anderen Datentypen zuweisen:

```
>>> i = 42
>>> i = "Hallo"
>>> i = [3, 9, 17]
```

Dennoch ordnet Python in jedem Fall einen speziellen Typ oder genauer gesagt eine spezielle Klasse der Variablen zu. Dieser Datentyp wird aber automatisch von Python erkannt. Diese automatische Typzuordnung, die auch während der Laufzeit erfolgen kann, bezeichnet man als „dynamische Typdeklaration". Mit der Funktion `type` können wir uns den jeweiligen Typ ausgeben lassen:

```
>>> i = 42
>>> type(i)
<class 'int'>
>>> i = "Hallo"
>>> type(i)
<class 'str'>
```

```
>>> i = [3, 9, 17]
>>> type(i)
<class 'list'>
```

Während sich bei statischen Typdeklarationen, also bei Sprachen wie C und C++, nur der Wert, aber nicht der Typ einer Variablen während eines Laufs ändert, kann sich bei dynamischer Typdeklaration, also in Python, sowohl der Wert als auch der Typ einer Variablen ändern.

Aber Python achtet auf Typverletzungen zur Laufzeit eines Programms. Man spricht von einer Typverletzung[9], wenn Datentypen beispielsweise aufgrund fehlender Zuweisungskompatibilität nicht regelgemäß verwendet werden. In Python kann es nur bei Ausdrücken zu Problemen kommen, da man ja einer Variablen einen beliebigen Typ zuordnen kann. Eine Typverletzung liegt beispielsweise vor, wenn man eine Variable vom Typ int zu einer Variablen vom Typ str addieren will. Es wird in diesem Fall ein TypeError generiert:

```
>>> x = "Ich bin ein String"
>>> y = 42
>>> z = x + y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Allerdings kann man Integer- und Float-Werte – auch wenn es sich um unterschiedliche Datentypen handelt – in einem Ausdruck mischen. Der Wert des Ausdrucks wird dann ein Float.

```
>>> x = 12
>>> y = 3.5
>>> z = x * y
>>> z
42.0
>>> type(x)
<class 'int'>
>>> type(y)
<class 'float'>
>>> type(z)
<class 'float'>
```

4.5 Typumwandlung

In der Programmierung bezeichnet man die Umwandlung eines Datentyps in einen anderen als Typumwandlung.[10] Man benötigt Typumwandlungen beispielsweise, wenn man Strings und numerische Werte mit dem Stringoperator „+" konkatenieren möchte.

[9] engl. type conflict
[10] engl. type conversion oder type cast

```
>>> first_name = "Henry"
>>> last_name = "Miller"
>>> age = 20
>>> print(first_name + " " + last_name + ": " + str(age))
Henry Miller: 20
```

In dem Beispiel haben wir den Wert von `age` explizit mit der Funktion `str` in einen String gewandelt. Man bezeichnet dies als explizite Typumwandlung. Hätten wir in obigem Beispiel nicht den Integer-Wert `age` in einen String gewandelt, hätte Python einen TypeError generiert:

```
>>> print(first_name + " " + last_name + ": " + age)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Der Text der Fehlermeldung besagt, dass Python keine implizite Typumwandlung von `int` nach `str` vornehmen kann. Prinzipiell unterstützt Python keine impliziten Typumwandlungen, wie sie in Perl oder PHP möglich sind. Dennoch gibt es Ausnahmen so wie unser Beispiel, in dem wir Integer- und Float-Werte in einem Ausdruck gemischt hatten. Dort wurde der Integer-Wert implizit in einen Float-Wert gewandelt.

4.6 Datentyp ermitteln

In vielen Anwendungen muss man für ein Objekt bestimmen, um welchen Typ bzw. um welche Klasse es sich handelt. Dafür bietet sich meistens die eingebaute Funktion `type` an. Man übergibt ihr als Argument einen Variablennamen und erhält den Typ des Objekts zurück, auf das die Variable zeigt:

```
>>> l = [3, 5, 4]
>>> type(l)
<class 'list'>
>>> x = 4
>>> type(x)
<class 'int'>
>>> x = 4.5
>>> type(x)
<class 'float'>
>>> x = "Ein String"
>>> type(x)
<class 'str'>
```

Als Alternative zur Funktion `type` gibt es noch die eingebaute Funktion `isinstance`, die einen Wahrheitswert „True” oder „False” zurückgibt.

```
isinstance(object, ct)
```

`object` ist das Objekt, das geprüft werden soll, und `ct` entspricht der Klasse oder dem Typ, auf den geprüft werden soll. Im folgenden Beispiel prüfen wir, ob es sich bei dem Objekt s um ein String handelt:

```
>>> s = "Ich bin ein String"
>>> isinstance(s, str)
True
```

In Programmen kommt es häufig vor, dass wir für ein Objekt wissen wollen, ob es sich um einen von vielen Typen handelt. Also zum Beispiel die Frage, ob eine Variable ein Integer oder ein Float ist. Dies kann man mit der Verknüpfung `or` lösen:

```
>>> x = 4
>>> isinstance(x, int) or isinstance(x, float)
True
>>> x = 4.8
>>> isinstance(x, int) or isinstance(x, float)
True
```

Allerdings bietet `isinstance` hierfür eine bequemere Möglichkeit. Statt eines Typs gibt man als zweites Argument ein Tupel von Typen an:

```
>>> x = 4.8
>>> isinstance(x, (int, float))
True
>>> x = (89, 123, 898)
>>> isinstance(x, (list, tuple))
True
>>> isinstance(x, (int, float))
False
```

Möchte man in einem Programm auf einen bestimmten Typ prüfen, so kann man obiges Vorgehen wählen, aber häufig lässt sich dies eleganter mit Ausnahmehandlungen lösen, die wir in Kapitel Ausnahmebehandlung, Seite 215 behandeln.

5 Sequentielle Datentypen

5.1 Übersicht

Auch andere Programmiersprachen wie beispielsweise Perl und Java kennen Datentypen für Strings und Listen, aber die Leistung von Python besteht darin, dass man solche Datentypen zu einer logischen Basisklasse „Sequentielle Datentypen" zusammenfasst. Eines haben nämlich Strings, Listen und Tupel gemeinsam: Es handelt sich jeweils um Datentypen, deren Elemente fortlaufend, also sequentiell, angeordnet sind. Bei Strings handelt es sich dabei um gleichartige Objekte, d.h. Zeichen wie Buchstaben, Ziffern, Sonderzeichen, und bei Listen oder Tupel handelt es sich um beliebige Objekte.

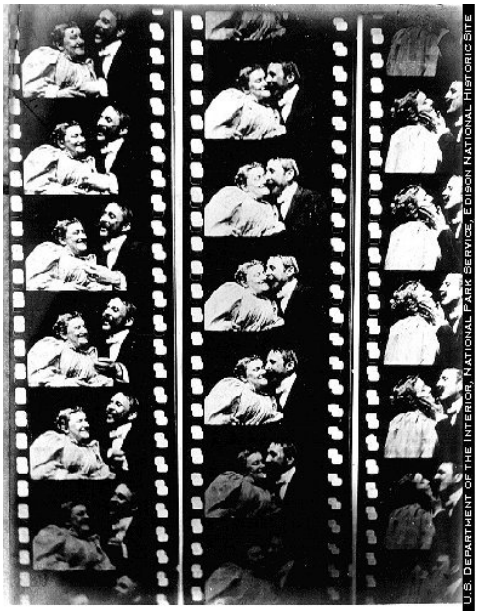


Bild 5.1 Fotos aus dem Film „The Kiss" von 1896

In Python gibt es deshalb gleichnamige Methoden oder Zugriffsmethoden für sequentielle Datentypen, wenn eine solche Operation für einen Typ möglich ist. Beispielsweise kann man, wie wir in diesem Kapitel kennenlernen, mit obj[0] auf das erste Element eines sequentiellen Datentyps zugreifen, egal ob es sich um einen String, eine Liste oder ein Tupel handelt. Aber man kann obj[0] nicht verändern, falls `obj` ein String ist.

Python stellt die folgenden sequentiellen Datentypen zur Verfügung: