Python Kompendium

Ein umfassender Einstieg in die Programmierung mit Python 3

Sarah Schmitt

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

www.bmu-verlag.de
info@bmu-verlag.de

Lektor: Matthias Kaiser
Einbandgestaltung: Pro ebookcovers Angie
Druck und Bindung: Wydawnictwo Poligraf sp. zo.o. (Polen)

Taschenbuch-ISBN: 978-3-96645-021-8
Hardcover-ISBN: 978-3-96645-049-2
E-Book-ISBN: 978-3-96645-020-1

Python Kompendium

Inhaltsverzeichnis

Über die Autorin

Sarah Schmitt studierte Mathematik und Philosophie in Heidelberg, Neu-Delhi und Berlin. Seit 2014 ist sie freiberuflich in den Bereichen Datenanalyse, Statistik und Algorithmen tätig. Darüber hinaus arbeitet sie als Übersetzerin, Lektorin und Autorin in mehreren Sprachen. Sie reist gerne und lebt in Berlin.

Widmung

Ich widme dieses Buch allen Frauen, die programmieren lernen wollen oder es bereits können. Weitermachen!

Danksagung

Ich danke meinen Freundinnen Doina Proorocu, Mahya & Mahsa Ilaghi und Jessica de Jesus de Pinho Pinhal für die wertvollen fachlichen Ratschläge, die stets zur richtigen Zeit kamen, sowie Albrecht Werner für das ebenso wertvolle Lektorat. Den eben genannten samt Yasmina Zian, Rahim Abdullayev, Hannelore Besser und Katie Pope danke ich für die freundschaftliche Unterstützung während der vielen Monate, die ich an diesem Buch schrieb. Ohne euch hätte ich es nicht geschafft!

Kapitel 1

Einleitung

Es macht Spaß, eine Programmiersprache zu erlernen. Die Fähigkeit, programmieren zu können, kann Ihnen eine ganz neue, faszinierende Welt eröffnen. Ideen, die Sie eventuell schon länger mit sich getragen haben, werden Sie effizient in die Tat umsetzen können, um damit nützliche Ergebnisse zu erzielen.

Ganz gleich, ob Sie für Ihre berufliche Weiterentwicklung programmieren lernen wollen oder Programme schreiben möchten, um damit Aufgaben des Alltags und eigene Projekte mithilfe neuer Methoden präzise und zeitsparend zu meistern – Programmierkenntnisse können Ihnen dabei helfen, Ihr analytisches Denkvermögen zu erweitern und gänzlich neue Wege aufzeigen, die Sie zuvor womöglich nicht gesehen haben.

Indem Sie lernen, die immanente Logik unterschiedlicher Problemstellungen zu erfassen und diese anschließend in strukturierter Form durch eine formalisierte Sprache auszudrücken, können Sie konkrete Lösungen für spezifische Aufgaben erzeugen. Womöglich mag Ihnen diese neue Denk- und Herangehensweise anfangs befremdlich oder sogar unnatürlich vorkommen. Sie werden jedoch schnell feststellen, dass sie zusätzlich zur intellektuellen Herausforderung ebenfalls ganz praktische Vorteile bringt und Ihnen dabei einen großen Freiraum an Kreativität lässt.

Da die Digitalisierung mittlerweile Einzug in die meisten unserer Lebens- und Arbeitsbereiche gehalten hat, ist programmiertechnisches Know-how in der heutigen Zeit sehr förderlich. Die Nachfrage nach Fachkräften mit dem entsprechenden Wissen wächst stetig, und zwar nicht nur im IT-Bereich: In den unterschiedlichsten Berufsfeldern sind solide Programmierkenntnisse mittlerweile eine gern gesehene Zusatzqualifikation und mitunter sogar eine wesentliche Voraussetzung für eine Anstellung. Die Entscheidung, eine Programmiersprache zu erlernen, ist oftmals bereits der erste Schritt in eine zeitgemäßere Arbeitswelt.

In diesem Buch werden wir die Programmiersprache *Python* ausführlich behandeln. Python ist einerseits leicht zu erlernen, lässt sich jedoch gleichzeitig sehr vielseitig und fachspezifisch einsetzen. Im Vergleich zu älteren Programmiersprachen wie *C* handelt es sich bei Python um eine neuere Sprache, in der moderne Programmiermethoden implementiert sind. Aus diesem Grund bietet Python zusätzlich eine gute Voraussetzung, um weitere Programmiersprachen schneller zu erlernen.

Die Sprache erfreut sich insbesondere seit einigen Jahren immer größer werdender Beliebtheit und wird von vielen großen Unternehmen und Institutionen innerhalb

von Wirtschaft, Forschung und Wissenschaft eingesetzt. So wurde beispielsweise die Google-Plattform mithilfe von Python entwickelt, und auch die Weltraumbehörde NASA setzt Python für ihre Berechnungen ein. Mit Python lassen sich Webseiten erstellen, komplexe Berechnungen effizient umsetzen, Datenanalysen vornehmen, Spiele kreieren, graphische Benutzeroberflächen erstellen, E-Mails versenden und vieles mehr.

Selbst, wenn Sie keinen technischen Hintergrund haben, ist es also eine gute Idee, mit Python in die Welt des Programmierens einzusteigen. In diesem Kapitel werden wir Ihnen zunächst einen kurzen Einblick darüber geben, was Python ist und welche Vorteile diese Sprache bietet, bevor wir in den darauffolgenden Kapiteln ihre Funktionsweise genauer kennen lernen.

1.1 Über dieses Buch

Dieses Buch will Anfänger und Menschen, die bereits Erfahrungen mit anderen Programmiersprachen gesammelt haben, gleichermaßen ansprechen. Es stellt eine praktische Einführung in die Programmierung mit Python dar und will fundiertes Wissen vermitteln, ohne sich dabei in allgemeinen, allzu theoretischen Konzepten des Programmierens zu verlieren. Auch Anfänger ohne jegliche mathematischen oder programmiertechnischen Vorkenntnisse sollten keinerlei Schwierigkeiten haben, den Inhalten dieses Buches zu folgen.

Als Einstieg in die Programmierung mit Python empfiehlt sich in jedem Fall eine gründliche Bearbeitung der Kapitel 1 bis 12. Für einen umfassenderen Einblick in Python können Sie danach sukzessive mit den nachfolgenden Kapiteln fortfahren, oder – je nach Schwerpunkt und Interessengebiet – einzelne für Sie relevante Themenbereiche aus den Kapiteln 13 bis 18 auswählen. Die Themengebiete werden dabei ausgehend von ihren grundlegenden Bausteinen bis in eine differenziertere Tiefe vorgestellt und ausführlich behandelt. Neu eingeführte Begriffe werden direkt an Ort und Stelle erklärt.

Ab Kapitel 4 sind die Kapitel zusätzlich mit vielen Beispielen und Übungsaufgaben versehen, anhand derer Sie das Gelernte wiederholen und verfestigen können. Alle Programmbeispiele aus dem Buch finden Sie zudem als Download unter https://bmu-verlag.de/books/python-kompendium/. Wir ermutigen Sie jedoch stets dazu, auf eigene Faust Beispiele zu verfassen und diese zu testen, anstatt lediglich die Beispiele in diesem Buch zu kopieren. Dadurch erlangen Sie sogleich mehr Sicherheit und eine spielerische Handlungsfreiheit im Umgang mit Python.

Menschen, die bereits Vorkenntnisse in anderen Programmiersprachen gesammelt haben, werden eventuell an einigen Stellen Inhalte wiedererkennen oder gewisse Ähnlichkeiten zu bereits erworbenem Wissen feststellen. Sie können insbesondere

die Einleitungen zu den einzelnen Kapiteln überfliegen. Wir empfehlen diesen Lesenden dennoch, die Themen aufmerksam durchzuarbeiten, da diese in der Regel zügig über das Grundwissen hinaus vertieft werden und somit auch für Menschen mit Vorkenntnissen eine wertvolle Wissenserweiterung bieten.

Wir haben versucht, geschlechtsspezifische Formulierungen weitestgehend zu umgehen und stattdessen neutrale Geschlechtsbezeichnungen zu verwenden. An einigen wenigen Stellen (etwa: „Anfänger“ oder „Nutzer“), wo dies die Einfachheit des Leseflusses unnötig verkomplizieren würde, verwenden wir jedoch ausschließlich die männliche Form. Frauen und Menschen anderer oder keiner geschlechterspezifischen Orientierung sollten dies keineswegs als Nichtbeachtung ihrer Präsenz verstehen und können sich beim Lesen in jedem Fall gleichermaßen angesprochen fühlen.

Es ist sicherlich sinnvoll, bei der Arbeit mit Python im Hinterkopf zu behalten, dass es im Internet zahlreiche Community-Plattformen und Foren für Python gibt, in denen man im Zweifelsfall Fragen stellen, über verschiedene Themen nachlesen oder eigene Beiträge verfassen kann. Spätestens dann, wenn man das erste eigene, größere Programm schreiben möchte, kommt man meistens nicht umhin, im Internet nach weiteren Informationen oder Ratschlägen zu suchen. Sie sollten es sich daher so früh wie möglich zur Gewohnheit machen, bei Bedarf selbstständige Recherchen im Internet anzustellen.

Einige der größten Wissens-Plattformen für Python sind:

www.python-forum.de/ (deutschsprachig)
www.stackoverflow.com (*die* Internetplattform für Softwareentwicklung)
www.python.org (die offizielle Python-Webseite)
www.pypi.org (*Python Package Index*, Sammlung verschiedener Pakete)
https://pyslackers.com/ (eine Webseite der Python-Community)

Online-Material sollte man dabei immer mit einer Prise Skepsis begegnen. Denn oftmals beeinträchtigt die schiere Menge an Informationen im Internet aufgrund ihrer leichten Zugänglichkeit und Erzeugbarkeit gleichzeitig die Qualität der Beiträge, sodass das Heranziehen von Fachliteratur in Form von Büchern, insbesondere bei fortgeschritteneren Themen, dennoch äußerst ratsam und vorteilhaft ist.

An dieser Stelle sei auch erwähnt, dass die englischsprachige (Online-)Literatur, wie in der gesamten IT-Welt, am umfangreichsten und detailliertesten ist, und dass es sich lohnt, wenn nicht sogar unumgänglich ist, sich von Anfang an parallel zur deutschen auch die englische Terminologie anzueignen. In diesem Buch werden Sie die wichtigsten Begriffe daher stets ebenfalls in Klammern auf Englisch angegeben vorfinden.

Sollten Sie einmal in der Bearbeitung dieses Buches nicht weiterkommen – vielleicht, weil Ihnen bestimmte Inhalte unverständlich waren oder Sie den Überblick verloren

haben – so nehmen Sie es mit Gelassenheit. Legen Sie eine Pause ein, machen Sie einen Spaziergang, schauen Sie sich einen Monty-Python-Sketch an, beschäftigen Sie sich mit etwas Anderem. Zu einem späteren Zeitpunkt können Sie sich wieder entspannt und mit klarem Kopf an die Lektüre setzen.

Wir wünschen viel Spaß und Freude beim Lernen!

1.2 Python: eine einfach zu erlernende Programmiersprache

Python ist eine flexible, einfache und elegante Programmiersprache, die sowohl für Anfänger als auch Fortgeschrittene bestens geeignet ist. Die klare, übersichtliche Syntaxstruktur macht es insbesondere Anfängern vergleichsweise leicht, mit Python den Einstieg in das Programmieren zu meistern.

In Bezug auf Programmiersprachen bezeichnet die *Syntax* – ähnlich wie bei natürlichen Sprachen – die den Satzbau regelnden formalen Vorschriften. Diese geben an, welche Kombinationen aus Symbolen (Zeichenketten, Satzzeichen, Sätzen, etc.) eine korrekte Struktur innerhalb der jeweiligen Sprache sind. Syntaxfehler sind dann beispielsweise Symbole, die an der falschen Stelle stehen oder Rechtschreibfehler enthalten. Ebenfalls als Syntaxfehler einzuordnen sind Wörter oder Befehle, die die Programmiersprache nicht kennt oder auch Ausdrücke, die in einem falschen Kontext verwendet werden. Enthält der Programmcode Syntaxfehler, lässt sich das Programm nicht zum Laufen bringen. Syntaxfehler passieren häufig und können selbst erfahrenen Programmierern und Programmiererinnen unterlaufen. Die Suche nach der fehlerhaften Stelle ist dabei oftmals mühsam und zeitaufwändig, da der Fehler zumeist nicht in den eigentlichen Befehlen liegt, sondern beispielsweise bloß durch eine fehlende Klammer, einen fehlenden Doppelpunkt oder einen Rechtschreibfehler verursacht wurde. Gerade Neulinge tun sich oftmals schwer damit, die Fehler in der Syntax zu finden – und können ihr Programm dann nicht starten.

Python überzeugt hier durch gute Lesbarkeit und ein ordentliches visuelles Layout. So kommt diese Programmiersprache beinahe gänzlich ohne Klammern aus: Zusammengehörende Blöcke werden anstatt durch Satzzeichen (wie z. B. Klammern) oder Befehlswörter bloß durch Einrückungen im Code ausgewiesen und sogleich sinnvoll gruppiert. Während in vielen anderen Programmiersprachen zudem verlangt wird, dass auf jeden Befehl ein Semikolon folgt, verzichtet Python auf den Einsatz von Semikolons nach Befehlen. All dies reduziert die Fehleranfälligkeit beim Schreiben von Programmcode erheblich und trägt zu Pythons Benutzerfreundlichkeit bei.

Wenn Sie sich bereits mit Programmiersprachen beschäftigt haben, so wissen Sie, was eine *Variable* ist. Ähnlich wie Variablen in der Mathematik enthalten Variablen beim Programmieren bestimmte Informationen, auf die später mittels eines Variablennamens zugegriffen werden kann, um damit weitere Operationen und Berech-

nungen auszuführen. Variablen können dabei nicht nur aus unterschiedlichen Zahlentypen (ganzen Zahlen, Fließkommazahlen, etc.), sondern ebenso aus Buchstaben (sog. *Strings* bzw. *Zeichenketten*) oder auch anderen, komplexeren Datenstrukturen bestehen. In einigen Programmiersprachen müssen Variablen anfangs deklariert und mit einem festen, unveränderlichen Datentyp versehen werden. Dies kann mitunter zu Fehlern führen, wenn man die Deklaration am Anfang vergisst oder versucht, mit verschiedenen Variablentypen zu arbeiten, die per Definition inkompatibel sind.

Python ist hier anders. Anwendende haben viele Freiheiten beim Gebrauch unterschiedlicher Variablentypen, da diese anfangs nicht deklariert werden müssen – der jeweilige Variablentyp wird dynamisch vergeben und kann auch später noch innerhalb des Programms verändert werden. Diese spezielle Verwendung der Variablen in Python nennt man *dynamische Typisierung*.

All diese Eigenschaften tragen dazu bei, dass Python einfach zu erlernen ist und auch auf fortgeschrittenem Niveau Spaß in der Anwendung macht.

1.3 Die Entwicklungsgeschichte

Python wurde Anfang der 90er Jahre vom niederländischen Programmierer Guido von Rossum am *Centrum Wiskunde & Informatica* in Amsterdam entwickelt. Über seine Beweggründe, eine neue Programmiersprache zu entwickeln, schrieb von Rossum im Jahr 1996 Folgendes:

> *„Vor über sechs Jahren, im Dezember 1989, suchte ich nach einem ‚Hobby'-Programmierprojekt, das mich über die Weihnachtswoche beschäftigen würde. Mein Büro [...] in Amsterdam würde geschlossen bleiben, aber ich hatte zuhause einen PC und sonst nicht viel zu tun. Ich beschloss, einen Interpreter für die neue Skriptsprache zu schreiben, über die ich in letzter Zeit nachdachte: Ein Nachfolger von ABC, der auch Unix- und C-Hacker ansprechen würde. Ich wählte Python als Arbeitstitel für das Projekt, da ich in einer leicht respektlosen Stimmung (und ein großer Fan des Monty Python's Flying Circus) war."*
>
> – frei übersetzt aus der Einleitung von *Programming Python*
> [Autor: Mark Lutz; O'Reilly Verlag]

Im Jahr 1994 erschien mit Python 1.0 die erste Vollversion. In den darauffolgenden Jahren kamen zahlreiche weitere Versionen heraus, die den Funktionsumfang von Python erheblich ausbauten, bis im Jahr 2000 sodann Python 2.0 erschien. Diese Version enthielt erstmals die sogenannte *Garbage Collection* (engl. für *Müllabfuhr*), bei der der Speicherplatz, den ein Programm besetzt, automatisch belegt und wieder freigegeben wird, sodass die Programmiererin oder der Programmierer die Speicherplatzverwaltung nicht manuell vornehmen muss – was wiederum Zeit spart und den

Programmiervorgang erleichtert. Bemerkenswert an Python 2.0 war auch der Entwicklungsprozess an sich, in dem Transparenz und die Mitwirkung der Community eine große Rolle spielten.

Nach einer langen Testphase erschien im Dezember 2008 schließlich Python 3.0, auch „Python 3000" oder „Py3K" genannt. Diese dritte Version war eine komplette Überarbeitung der Sprache, in der viele Unstimmigkeiten und Designfehler behoben wurden. Dabei musste bei der Entwicklung jedoch auf die Option der Abwärtskompatibilität verzichtet werden. Das bedeutet, dass Programme, die mit Python 2 geschrieben wurden, nicht mit einem für Python 3 konzipierten Interpreter ausgelesen werden können. Python 2.0 wurde bis zur Version 2.7, die im Jahr 2010 herauskam, zwei Jahre lang parallel zur Version 3.0 weiterentwickelt. In dieser Zeit gab es daher stets zwei aktuelle Python-Versionen, zwischen denen sich die Anwender entscheiden mussten. Obwohl ältere Programme häufig noch in Python 2 geschrieben sind, wird für neuere Programme in der Regel Python 3 verwendet. Darüber hinaus finden sich noch viele ältere Lehrbücher zu Python 2. Der Support und Updates für Python 2 wurden ab 1. Januar 2020 jedoch eingestellt.

In diesem Buch werden wir uns dem aktuellen Stand entsprechend ausschließlich mit Python 3 befassen, sodass kein Wissen über die genauen Unterschiede zwischen Python 2 und Python 3 vonnöten sein wird. Sie sollten sich lediglich der Möglichkeit bewusst sein, dass Code aus anderen Quellen mitunter nicht kompatibel zur aktuellen Version sein könnte. Beim Verwenden von fremdem Code ist es daher stets ratsam, vorab zu überprüfen, in welcher Version von Python dieser geschrieben wurde.

1.4 Die Ziele bei der Entwicklung von Python

Wie bereits erwähnt, wurde Python ursprünglich als Nachfolger für die Programmier-Lehrsprache *ABC* entwickelt. Im Vergleich zu ABC sollte Python überschaubarer und leicht zu erweitern sein. Gute Lesbarkeit und Einfachheit standen dabei an erster Stelle. So beschränkt die reduzierte Syntax beispielsweise den Programmcode auf dessen wesentliche Elemente, sodass diese im Vordergrund bleiben.

In der 2004 erschienenen Reihe *The Zen of Python* wird die minimalistische Philosophie dieser Programmiersprache in knappen, humoristischen Aphorismen dargelegt. Das Werk enthält Leitsätze wie:

„Schön ist besser als hässlich.
Explizit ist besser als implizit.
Einfach ist besser als komplex.
Komplex ist besser als kompliziert.
Lesbarkeit zählt."

– Tim Peters, *The Zen of Python* (frei übersetzt)

Darüber hinaus sollte Python intuitiv und genauso einfach zu verstehen sein wie die englische Sprache; die Anwendung sollte Spaß machen. Dies verdeutlicht nicht zuletzt die originelle Namensgebung: Python ist nicht, wie man zunächst vermuten könnte, nach der gleichnamigen Würgeschlangenart benannt, sondern nach der britischen Fernsehsendung *Monty Python's Flying Circus*. Obwohl inzwischen tatsächlich eine Schlange als Symbol der Programmiersprache dient, ist die ursprüngliche Bedeutung des Namens noch an vielen Stellen erkennbar. So sind die Webseite der Python-Dokumentation und viele andere Lehrbücher über Python mit Zitaten aus Monty-Python-Sketchen versehen, und Anspielungen auf die Fernsehsendung im Code werden gerne eingesetzt.

Der eigentliche Programmcode ist in Python sehr knappgehalten. Die gleichen Programme mit denselben Funktionen, in anderen Sprachen verfasst, benötigen dort in der Regel mehr Zeilen als in Python. Anfänger sind hier also im Vorteil: übersichtliche, kürzere Codebeispiele sind leichter zu lesen und zu verstehen. Auch die Zeit, die zum Schreiben eines neuen Programms benötigt wird, bleibt hierdurch vergleichsweise gering. Aus diesem Grund wird Python gerne für Projekte mit hohem Zeitdruck eingesetzt.

Darüber hinaus verfügt Python über viele weitere Eigenschaften, die zur Übersichtlichkeit und leichten Verständlichkeit des Codes beitragen. Im Vergleich zu anderen Programmiersprachen kommt Python etwa nur mit sehr wenigen grundlegenden Schlüsselwörtern aus. Die Standardbibliothek, eine von Pythons größten Stärken, wurde bewusst überschaubar gehalten und lässt sich anschließend leicht erweitern. Das Potential der Sprache wird somit keineswegs durch diese anfängliche Einschränkung beeinträchtigt. Für Neulinge hat dies jedoch den Vorteil, dass sie anfangs vergleichsweise wenige Begriffe auswendig lernen müssen und sich leichter in die neue Sprache einarbeiten können.

Wichtig zu erwähnen ist außerdem, dass es sich bei Python um ein Open-Source-Projekt handelt. Hiermit wird ein Modell innerhalb der Softwareentwicklung bezeichnet, das um 2000 mit der zunehmenden Verbreitung des Internets entstand. *Open Source* bedeutet, dass die Öffentlichkeit den Quellcode (einer Sprache oder eines Programms) – im Gegensatz zu urheberrechtlich oder lizenzgeschützten Programmen – kostenlos einsehen, modifizieren oder weiterverbreiten kann. Open-Source-Projekte fördern infolgedessen die freiwillige Unterstützung von Anwendern, die eventuelle Verbesserungen oder Problemlösungen innerhalb der Community teilen können und somit zur Weiterentwicklung des Projekts beitragen. Pythons Einfachheit und Vielseitigkeit ließen so neben einer hohen Nutzerschaft auch sehr schnell eine große, aktive Community entstehen.

Die oberste Entscheidungsmacht über die Entwicklung von Python lag seit dessen Entstehung jedoch weiterhin bei Guido van Rossum als sog. *Benevolent Dictator for*

Life („wohlwollender Diktator auf Lebenszeit“). Obwohl er sich im Juli 2018 von dieser Position verabschiedete, beteiligt er sich nach wie vor an der Weiterentwicklung von Python.

1.5 Interpretiert vs. kompiliert

Bevor Sie die für das Programmieren mit Python nötige Software installieren und kennen lernen, ist es eine gute Idee, über die unterschiedlichen Implementierungsmöglichkeiten von Programmen Bescheid zu wissen. Dazu wollen wir ein wenig auf die internen Abläufe eingehen, die beim Ausführen eines Programms stattfinden.

Computerprogramme werden in gewöhnlichem Text geschrieben. Anders als natürliche Sprachen bestehen sie jedoch aus fest vorgegebenen Symbolen und Schlüsselbegriffen, die einer ebenfalls fest vorgegebenen Struktur folgen – der Syntax (Kapitel 1.2.). Diese Begriffe und Symbole sind sehr stark an die englische Sprache und an mathematische Symbole angelehnt, sodass selbst Anfänger mit grundlegenden Englisch- und Mathematikkenntnissen beim Lesen eines Computerprogramms oftmals bereits erahnen können, welche Aufgaben es ausführen soll.

Auf einer weniger abstrakten Ebene haben Programmabläufe jedoch eine ganz andere Gestalt. Zu Anfang bestehen alle Informationen nämlich aus reinem *Binärcode*. Das bedeutet, dass jede Information lediglich zwei verschiedene Zustände annehmen kann. Auf der Hardwareebene handelt es sich dabei in der Regel ursprünglich um elektrische Impulse mit den beiden möglichen Zuständen „niedrig“ und „hoch“, die im Rechner stellvertretend mit den Zahlen 0 und 1 bezeichnet werden. Jede Null und jede Eins ist ein sogenanntes *Bit* (von engl. ***b****inary dig****it***). Kombinationen dieser kleinsten Informationseinheiten werden als Wörter bezeichnet.

In modernen Computern kommen 64-Bit-Wörter zum Einsatz, wohingegen ältere Rechner mit 32-, 16- oder sogar 8-Bit-Wörtern arbeiteten. Jedes Wort, das im Prozessor ankommt, bewirkt eine bestimmte Ausgabe. Die Abfolge aller Wörter eines Programms führt somit dazu, dass der Prozessor das gewünschte Ergebnis ausgibt. Diesen aus Wörtern von Binärzahlen bestehenden Code nennt man *Maschinensprache* bzw. *Maschinencode*. Ein Beispiel für Maschinencode könnte etwa die folgende Form haben:

```
0000   11111010
0001   00110011
0002   11000000
0003   10001110
0004   11010000
0005   10111100
0006   00000000
0007   00000001
```

Höhere Programmiersprachen wie Python beruhen ebenfalls auf den Prinzipien der Maschinensprache, sind jedoch weitaus abstrakter und komplexer entwickelt. Damit ein in einer Hochsprache geschriebenes Computerprogramm ausgeführt werden kann, muss es zunächst in Maschinensprache übersetzt werden. Hierfür gibt es klassischerweise zwei Möglichkeiten:

Zum einen kann der Programmcode mithilfe eines *Compilers* in Maschinencode umgewandelt werden. Dies hat den Vorteil, dass bereits kompilierte Programme bei Folgeausführungen nicht erneut übersetzt werden müssen, was offensichtlich Zeit spart. Zudem überprüft der Kompilierer das Programm vor dessen Erstellung auf Fehler, sodass diese bereits davor behoben werden können. Der Nachteil an dieser Methode ist jedoch, dass die jeweilige Kompilation nicht plattformunabhängig ist und dass das Programm somit nicht ohne Weiteres auf verschiedenen Betriebssystemen laufen kann. Programmiersprachen wie *C* oder *C++* laufen beispielsweise über einen Compiler.

Eine andere Möglichkeit besteht darin, einen *Interpreter* zu verwenden. Ein Interpreter ist ein Programm, das den Code während seiner Ausführung einliest und ihn Zeile für Zeile direkt in Maschinensprache übersetzt. Ein interpretiertes Programm kann auf jedem Rechner laufen, der mit einem entsprechenden Interpreter ausgestattet ist, ohne dass das Betriebssystem des Rechners dabei eine Rolle spielt. Der Entwicklungsprozess gestaltet sich effizienter, da das Programm bei jedem erneuten Testen nicht aufs Neue kompiliert werden muss, sondern direkt mithilfe des Interpreters ausgeführt werden kann. Insbesondere bei größeren Programmen und mehreren hundert Kompiliervorgängen spart dies beim Entwickeln viel Zeit.

In Wirklichkeit schließen sich diese beiden Begriffe nicht gegenseitig aus, da jede Programmiersprache theoretisch betrachtet entweder interpretiert oder kompiliert werden kann. Während die Einteilung in *kompiliert* und *interpretiert* früher eine größere Rolle spielte, wird in modernen Programmiersprachen-Implementierungen zunehmend eine Kombination aus beiden Möglichkeiten innerhalb einer Plattform bevorzugt, sodass die einstige Unterscheidung weniger relevant ist. Heutzutage gibt es nur noch sehr wenige klassische Interpreter. Die meisten, darunter auch Python, kompilieren den Programmcode zunächst in einem Zwischenschritt in Bytecode, der anschließend durch einen Interpreter auf einer virtuellen Maschine (im Falle von Python von der *PVM*, ***P**ython **V**irtual **M**achine*) ausgeführt wird.

Kapitel 2

Die Vorbereitung

Bevor wir mit dem eigentlichen Programmieren anfangen können, müssen wir einige Vorbereitungen treffen. Wie Sie bereits aus dem vorigen Kapitel wissen, bestehen Computerprogramme aus reinem Text. Um diesen Text zu schreiben und ihn in einer Datei abzuspeichern, die vom Interpreter anschließend ausgelesen wird, benötigt man zunächst bloß ein entsprechendes Textverarbeitungsprogramm zum Erstellen von Textdokumenten – Microsoft Word oder LibreOffice sind Beispiele für derartige Software. Jedoch enthalten diese Programme zusätzlich zum Text noch weitaus mehr Informationen – etwa Angaben zur Schriftart, zur Schriftgröße sowie zu vielen anderen Formatierungsdetails –, die auf eine zum Programmieren ungeeignete Art und Weise gespeichert werden. Aus diesem Grund kommt beim Programmieren ein Texteditor zum Einsatz.

Auf Computern mit Windows als Betriebssystem ist bereits ein solcher Texteditor vorinstalliert: Notepad. Obwohl man mit dem Notepad-Editor theoretisch Programme schreiben könnte, ist dessen Funktionalität sehr begrenzt und nicht an die spezifischen Bedürfnisse des Programmierens angepasst. Nötig ist hier also eine Software mit umfangreicheren Möglichkeiten. Ein geeigneter Editor sollte unter anderem bestimmte Befehle automatisch erkennen und zur besseren Lesbarkeit farblich markieren können, selbstständig notwendige Einrückungen von zusammengehörenden Textblöcken vornehmen, über eine Suchen- und Ersetzen-Funktion verfügen und vieles mehr.

Um den geschriebenen Code sogleich ausführen und testen zu können, ist darüber hinaus ein Erstellungsprogramm notwendig. Zusätzlich sollte die Entwicklungssoftware über einen Debugger verfügen, der beim Programmieren dabei hilft, eventuelle Fehler im Skript, die das Programm an der Ausführung hindern, zu lokalisieren und zu beheben.

2.1 Die integrierte Entwicklungsumgebung

Eine *IDE* (*Integrierte Entwicklungsumgebung*, von engl. ***I**ntegrated **D**evelopment **E**nvironment*) ist dasjenige Tool, das all diese Anforderungen vereint. In einer IDE können Sie bequem Code verfassen, speichern, wiederholt testen und ausführen. Das Angebot an unterschiedlichen IDEs ist dabei schier unbegrenzt. Es gibt IDEs, die Sie für viele verschiedene Programmiersprachen verwenden können und solche, die nur jeweils eine Sprache unterstützen. Anwender werden schnell feststellen, dass der Aufbau der

meisten IDEs im Grunde recht ähnlich ist, während sie sich hinsichtlich ihres Designs, ihrer Funktionalität und Nutzerfreundlichkeit stark unterscheiden können.

Die Wahl der Entwicklungsumgebung sollte sich vor allem nach Ihren persönlichen Programmierzielen und Ihrem Interessenfeld richten. Mittlerweile gibt es sehr spezifische, mächtige Tools für die unterschiedlichsten Anwendungsbereiche, die mitunter bereits Pakete für spezifische Anwendungen beinhalten. Für Python stehen Ihnen über 50 solcher IDEs zur Verfügung, die meisten davon sind kostenfrei.

Anfänger haben zunächst die Möglichkeit, diejenige IDE zu verwenden, die bei der Installation des Python-Interpreters (unter Windows und macOS) automatisch mit enthalten ist: *IDLE* (engl. ***I**ntegrated **D**evelopment and **L**earning **E**nvironment*) ist ein einfach gehaltenes Tool, das besonders für Lernzwecke völlig ausreichend ist. Im Anschluss an die in den nächsten Abschnitten beschriebene Installation des Interpreters können Sie IDLE sofort testen und sehen, ob Sie gut damit zurechtkommen.

Im Hinblick auf fortgeschrittenere Ziele und Anforderungen, die auch Bestandteil dieses Buchs sein werden, wird IDLE jedoch womöglich nicht ausreichend sein. Einige der hierfür in Frage kommenden, empfehlenswerten IDEs sind etwa *Thonny* (für Anfänger; https://thonny.org/), *Visual Studio Code* (https://code.visualstudio.com/), *Spyder* (insbesondere für Datenanalyse; www.spyder-ide.org) oder *PyCharm* (www.jetbrains.com/pycharm). In den Beispielen dieses Buchs wird PyCharm zum Einsatz kommen. Hierbei handelt es sich um ein sehr beliebtes, komfortables und zugleich starkes Entwicklungstool.

Linux-Anwender verfügen in der Regel bereits über einen entsprechenden einfachen Texteditor, nicht jedoch über eine umfassende IDE. Bei Ubuntu ist dies *gEdit* und bei KDE *Kate*. Auch andere Distributionen sind meistens mit einem passenden Texteditor ausgestattet.

Im nächsten Abschnitt 2.2. werden wir erklären, wie Sie den Python-Interpreter, der den Code in Maschinensprache übersetzt, auf verschiedenen Betriebssystemen installieren. Im darauffolgenden Abschnitt 2.3. werden wir uns sodann genauer der Verwendung und Installation der PyCharm-IDE widmen und Ihnen in 2.4. zeigen, wie Sie diese anpassen können.

2.2 Den Python-Interpreter installieren

Wie bereits erwähnt, arbeitet Python mit einem Interpreter, der den großen Vorteil der Plattformunabhängigkeit bietet. Das bedeutet, dass Python-Code auf allen Betriebssystemen laufen kann, ohne dass dieser hierfür abgeändert werden muss. Den Python-Interpreter können Sie unter der folgenden Internetadresse herunterladen: https://www.python.org/downloads/.

Abb. 2.1 Die Startseite der Python-Entwicklergemeinschaft mit dem Download des Interpreters

Durch Anklicken der gelben Schaltfläche öffnet sich ein Fenster zum Download des Installations-Assistenten. In der Regel wird die Webseite automatisch die für das Betriebssystem Ihres Computers passende Version auswählen. Sollte dies nicht der Fall sein, können Sie die richtige Version auf derselben Seite auswählen. Hier finden sich zudem auch frühere Python-Versionen, darunter beispielsweise auch Python 2.7.

2.2.1 Unter Windows installieren

Python läuft erst ab Version 3.5 auf Windows Vista oder neueren Betriebssystemen.

Nachdem der Download beendet wurde, können Sie die Installation des Interpreters durch Anklicken der Datei starten. Das folgende Fenster wird nun angezeigt:

Abb. 2.2 Der Installations-Assistent für den Python-Interpreter

Hier müssen Sie darauf achten, dass Sie in der unteren Checkbox **Add Python 3.7 to PATH** einen Haken setzen. Dadurch wird es möglich sein, Python von der Eingabeaufforderung (Kommandozeile) aus zu starten, anstatt jedes Mal in das Verzeichnis wechseln zu müssen, in dem der Interpreter installiert wurde. Die eigenen Programme können somit nicht nur im Interpreter-Verzeichnis, sondern auch in anderen Verzeichnissen gespeichert und von dort aus aufgerufen werden. Mit **Install Now** bestätigen Sie anschließend die Installation, woraufhin der Python-Interpreter startklar ist.

Über das Startmenü können Sie nun überprüfen, ob Python korrekt installiert wurde. Wählen Sie hierzu unter der Liste der installierten Programme den Ordner **Python 3.7** und darin **Python 3.7 (32 Bit)** aus. Der Python-Interpreter öffnet sich in der Eingabeaufforderung, die Versions- und Installationsdetails werden angezeigt. In dieser Menüauswahl befindet sich ebenfalls die zuvor erwähnte Entwicklungsumgebung IDLE.

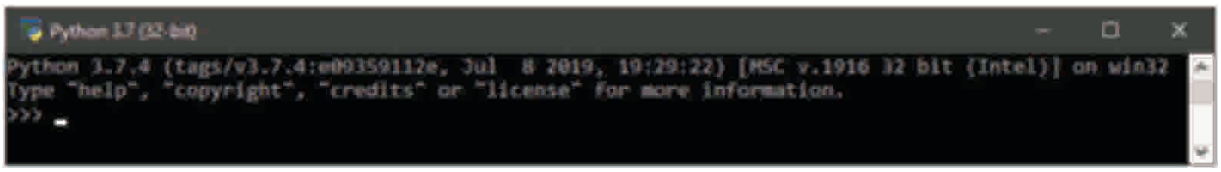

Abb. 2.3 Python-Installationsinfo im Python-Interpreter

Alternativ hierzu können Sie den Python-Interpreter auch direkt in der Eingabeaufforderung durch den Befehl `python` öffnen. Zur Eingabeaufforderung gelangen Sie

am schnellsten, indem Sie **cmd** in die Windows-Suche eingeben. Die folgende Ausgabe sollte angezeigt werden:

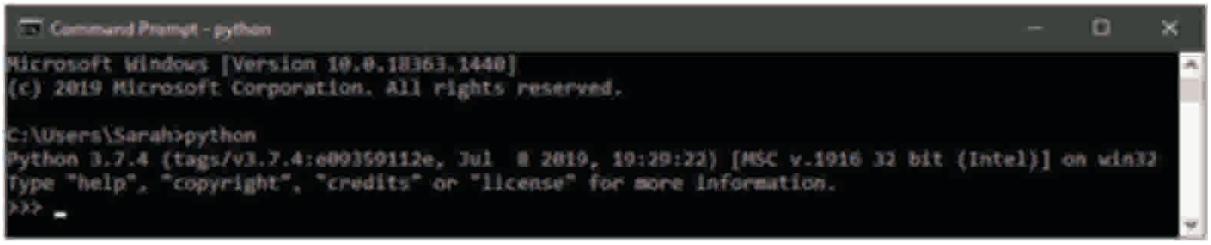

Abb. 2.4 Python-Installationsinfo

Die drei spitzen Klammern (>>>) bedeuten hier, dass der Python-Interpreter einsatzbereit ist und auf Befehle wartet.

In der Eingabeaufforderung – also nicht im Interpreter – können Sie übrigens jederzeit durch die Eingabe von `python --version` überprüfen, welche Python-Version auf Ihrem PC aktuell installiert ist.

2.2.2 Unter Linux installieren

Sehr wahrscheinlich ist auf Ihrem System bereits mindestens eine Version von Python vorhanden. Um zu überprüfen, ob es sich dabei um Python 2 oder um Python 3 handelt, können Sie in einem Terminal-Fenster (**Ctrl** + **Alt** + **T**) die Befehle `python -V` (für Python 2) bzw. `python3 -V` (für Python 3) eingeben. Die entsprechenden Installationsinformationen werden angezeigt. Um die derzeit aktuellste Python-Version, Python 3.7, zu verwenden, können Sie deren Installation durch den Befehl `sudo apt-get install python3.7` veranlassen. Mit dem Befehl `python3.7` lässt sich anschließend kontrollieren, ob die Installation richtig durchgeführt wurde.

2.2.3 Unter macOS installieren

Standardmäßig ist auf macOS Python 2.7 vorinstalliert. Um die aktuelle Version zu erhalten, müssen Sie die entsprechende Datei wie oben in 2.2. beschrieben herunterladen, ausführen und sodann den Schritten auf dem Bildschirm folgen. War die Installation erfolgreich, sollten auf die Eingabe von `python3` in einem Terminal-Fenster (zu finden unter **Applications → Utilities → Terminal**) hin die Installationsinformationen erscheinen.

2.3 PyCharm – Eigenschaften und Installation

Widmen wir uns nun dem Tool, mit dem wir im Rahmen dieses Buches hauptsächlich arbeiten werden: PyCharm. Zusätzlich zum Texteditor beinhaltet PyCharm weitere

Komponenten wie eine Quellcodeverwaltung, einen Debugger sowie Test- und Analysewerkzeuge. Hier sind einige der Vorteile, die der PyCharm-Texteditor gegenüber einem gewöhnlichen Texteditor wie Notepad bietet:

- Syntaxhervorhebung: Verschiedene Bestandteile des Codes werden automatisch erkannt und farblich markiert, sodass sie leichter voneinander zu unterscheiden sind.
- Zeilennummerierung: Durch die Nummerierung der Zeilen wird der geschriebene Programmcode übersichtlicher und man kann leichter zwischen verschiedenen Stellen navigieren. Zudem lassen sich Fehlermeldungen einfacher nachvollziehen, da diese normalerweise zusammen mit der Zeilennummer angeben werden.
- Code-Faltung: Einzelne zusammengehörende Blöcke lassen sich durch einen Klick auf das seitliche Minuszeichen ein- und wieder ausklappen. Das erhöht die Übersichtlichkeit beim Arbeiten mit dem Text.
- Suchen und Ersetzen-Funktion: Bestimmte Elemente im Code lassen sich gezielt suchen und durch einen anderen Befehl automatisch ersetzen. Dadurch lassen sich Korrekturen deutlich schneller ausführen.
- Automatische Vervollständigung von Befehlen: Sobald Sie die ersten Buchstaben eines Befehls eingetippt haben, öffnet sich ein Fenster, in dem Sie den richtigen Befehl unter verschiedenen Möglichkeiten auswählen können.
- Automatische Einrückung: Zusammengehörende Blöcke werden automatisch eingerückt. Da Einrückungen bei Python bestimmte Funktionen haben, ist diese Eigenschaft besonders praktisch.
- Kompilieren und ausführen: Die Programme können direkt vom Texteditor aus kompiliert und ausgeführt werden.

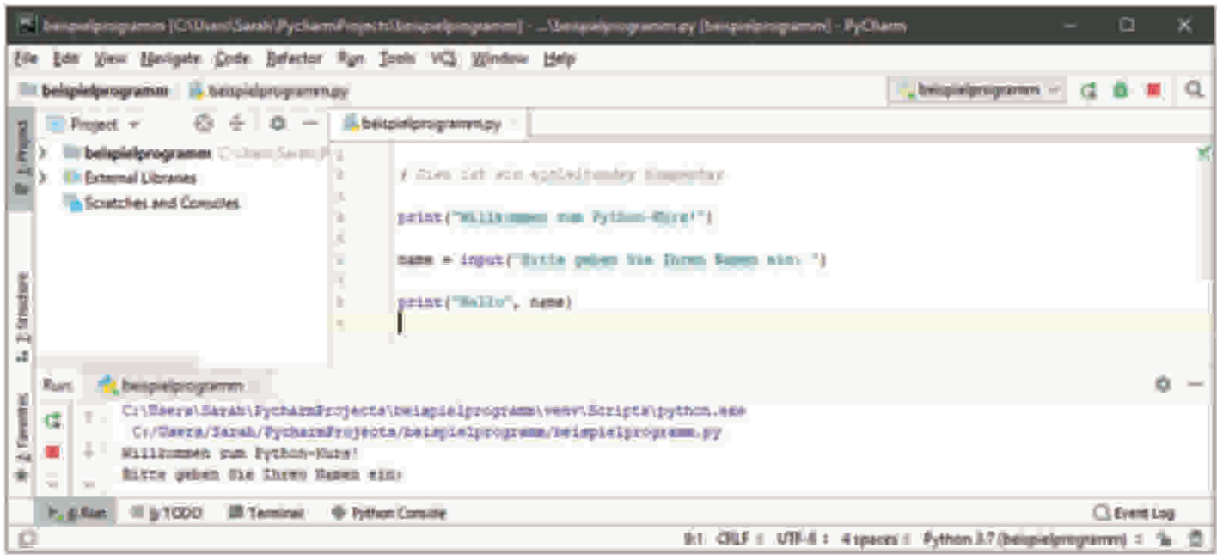

Abb. 2.5 Erste Ansicht der PyCharm-IDE

Sie können PyCharm auf der Seite http://www.jetbrains.com/pycharm/ für Ihr Betriebssystem herunterladen. Hier haben Sie die Auswahl zwischen zwei verschiedenen Versionen: der *Professional*-Version (zahlungspflichtig) und der quelloffenen *Community*-Version (kostenlos). Die Community-Version wird für unsere Zwecke vollkommen ausreichend sein.

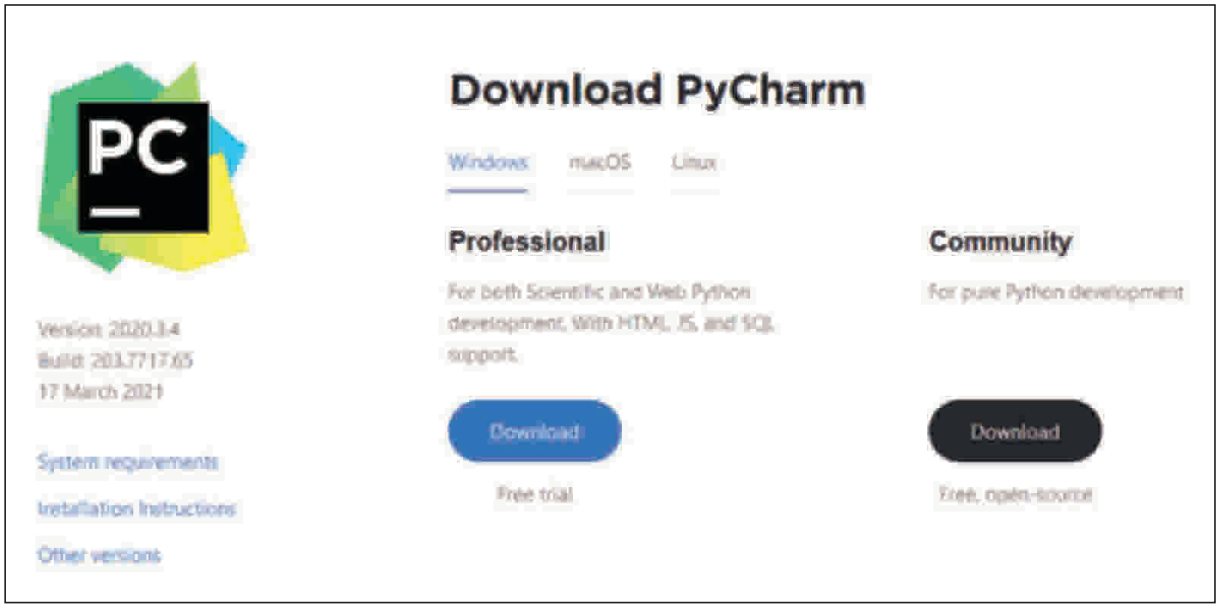

Abb. 2.6 Erhältliche PyCharm-Versionen

Die Wahl der Entwicklungsumgebung ist nicht zwingend. Sollten Sie feststellen, dass Sie mit einer anderen IDE besser zurechtkommen, können Sie für die Aufgaben und Beispiele in diesem Buch gerne auch diese verwenden. Eine Liste über einige allgemeine sowie speziell auf Python zugeschnittene IDEs samt Vor- und Nachteilen gibt es unter https://realpython.com/python-ides-code-editors-guide/. Eine vollständige Liste finden Sie unter https://wiki.python.org/moin/PythonEditors.

2.3.1 PyCharm: Installation unter Windows

Nachdem Sie die Installationsdatei heruntergeladen haben, folgen Sie den Schritten des Installationsprogramms. Wählen Sie zunächst den Ordner aus, in dem Sie PyCharm installieren möchten. Im darauffolgenden Fenster müssen Sie sodann angeben, ob Ihr Betriebssystem auf einer 32-Bit- oder einer 64-Bit-Architektur beruht. Dies können Sie in der Systemsteuerung unter **System → Systemtyp** herausfinden. Verwenden Sie tatsächlich ein 32-Bit-System, so müssen Sie zusätzlich unten im Fenster das Feld *Download and install JRE x86 by JetBrains* durch ein Häkchen aktivieren. Achten Sie in jedem Fall zudem darauf, dass Sie unter *Create Associations* einen Haken bei *.py* setzen. Dadurch werden Python-Quelltexte (also alle *.py*-Dateien) automatisch direkt mit PyCharm geöffnet. Wählen Sie zudem die Option *Add launchers dir to the PATH* aus.

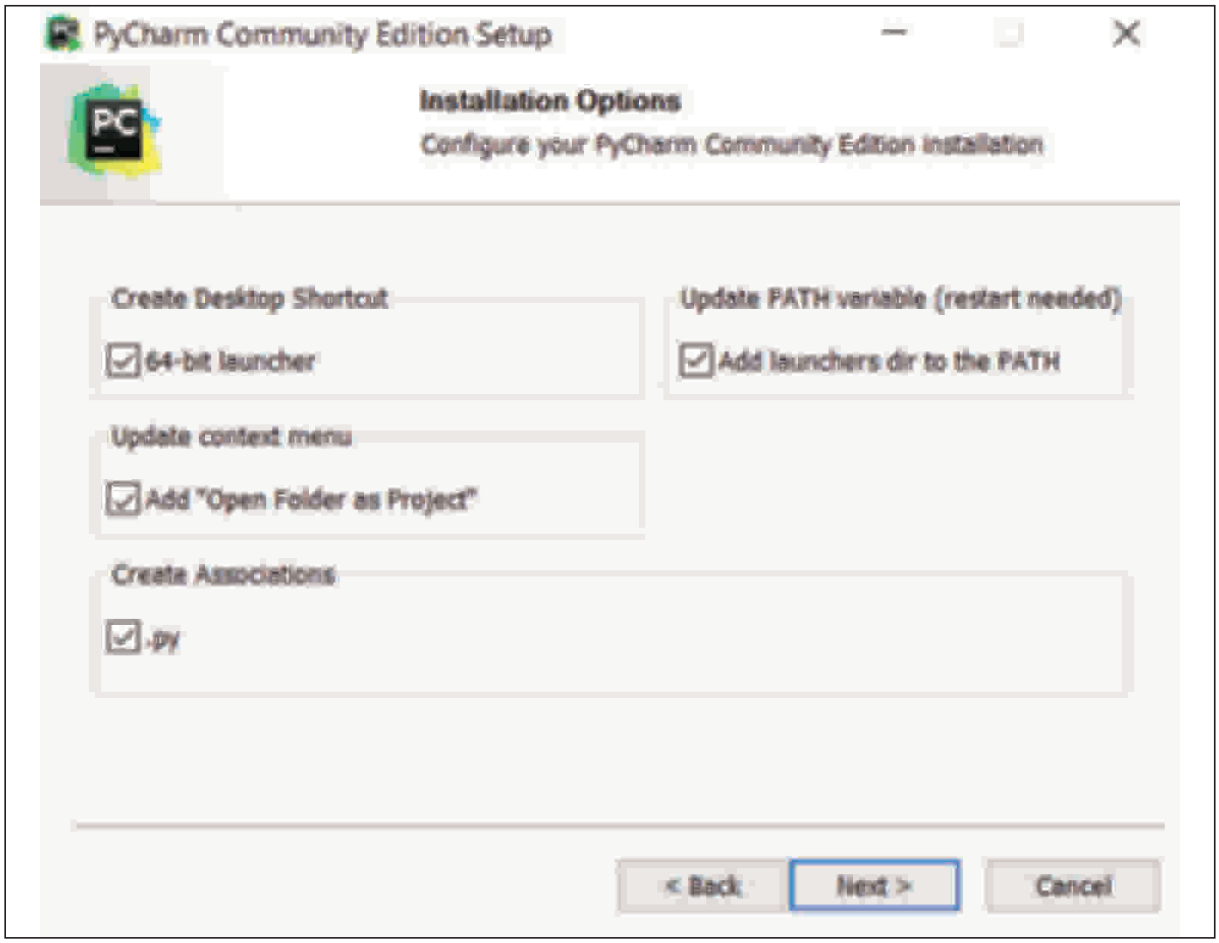

Abb. 2.7 PyCharm unter Windows installieren

2.3.2 PyCharm: Installation unter Linux

Ab Ubuntu 16.04 lässt sich PyCharm über ein Snap-Paket installieren. Durch den Kommandozeilenbefehl `sudo snap install pycharm-professional --classic` für die Professional-Version bzw. `sudo snap install pycharm-community --classic` für die kostenlose Community-Version können Sie die Installation veranlassen.

Unter Ubuntu 18.04 befindet sich PyCharm bereits in den offiziellen Paketquellen, was die Installation sehr einfach gestaltet. Klicken Sie im Dock auf **Ubuntu-Software** oder alternativ dazu auf **Anwendungen Anzeigen → Ubuntu-Software**. Geben Sie im Suchfeld oben rechts *pycharm* ein. Nun können Sie eine der angezeigten Versionen auswählen, beispielsweise die kostenfreie Version *PyCharm CE*. Bestätigen Sie Ihre Auswahl durch **Installieren** und starten Sie PyCharm.

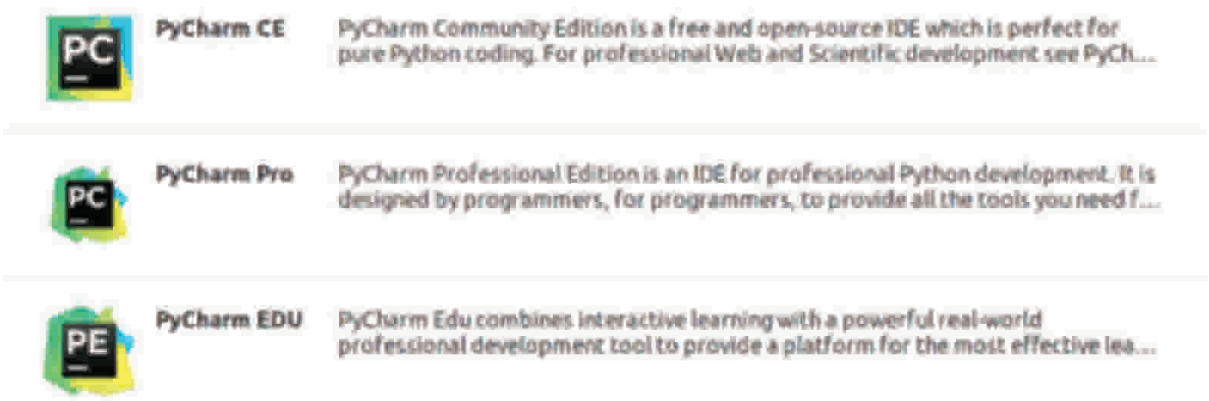

Abb. 2.8 Installation unter Ubuntu 18.04

Unter einer anderen Linux-Distribution sollten Sie den Anweisungen auf https://www.jetbrains.com/help/pycharm/installation-guide.html?section=Linux zur Installation als *.tar.gz*-Archiv folgen.

2.3.3 PyCharm: Installation unter macOS

Laden Sie die Installationsdatei unter der oben angegebenen Adresse herunter. Auch hier können Sie zwischen der Professional- und der Community-Version wählen. Öffnen Sie die *.dmg*-Datei nach dem Download. Anschließend öffnet sich ein Fenster. Nun müssen Sie das Icon *PyCharm CE* in den Ordner *Applications* ziehen. Anschließend können Sie das Image auswerfen und PyCharm aus Ihrem Applications-Ordner heraus starten.

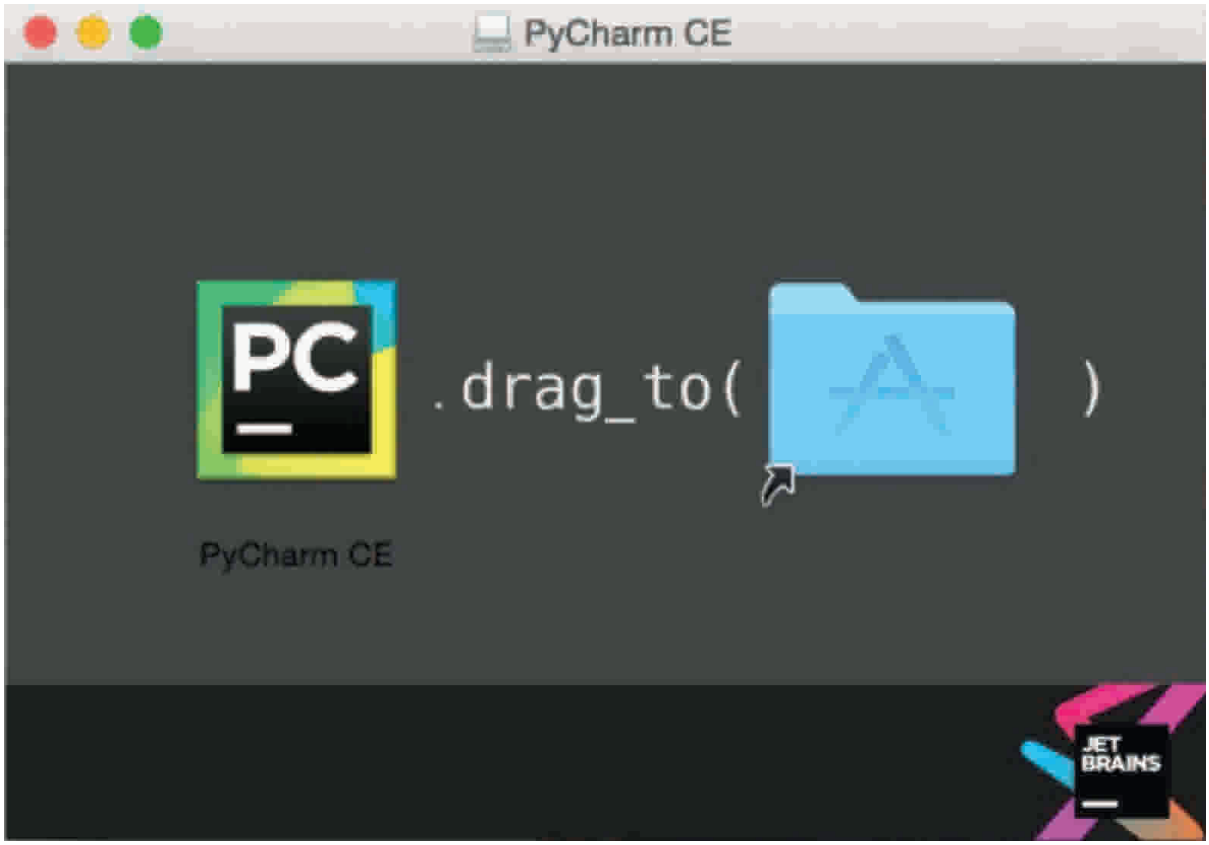

Abb. 2.9 PyCharm unter macOS installieren

2.4 PyCharm anpassen

Wenn Sie PyCharm zum ersten Mal starten, werden Sie gefragt, ob Sie bereits vorhandene Einstellungen einer alten Installation verwenden möchten. Klicken Sie **Do not import settings** und bestätigen Sie mit **OK**. MacOS-Nutzer können anschließend zwischen verschiedenen Tastaturlayouts wählen. Im nächsten Schritt können Sie ein Farbschema auswählen. Mit **Skip Remaining and Set Defaults** überspringen Sie die weiteren Einstellungen, die für uns nicht relevant sind. Ihre Einstellungen können Sie auch im Nachhinein jederzeit über das Einstellungsmenü einsehen und ändern. In das Einstellungsmenü gelangen Sie über das Hauptmenü (über **File → Settings...** bzw. macOS: **PyCharm → Preferences**). Hier müssen Sie nun den gewünschten Interpreter auswählen, den Sie zuvor installiert haben. Das geht über den Eintrag **Project → Project Interpreter** in der Auswahl auf der linken Seite. Anschließend sollte eine neue Auswahlbox mit dem installierten Interpreter erscheinen. Ist dieser nicht in der Liste enthalten, wählen Sie **Show All...** und die Schaltfläche mit dem Plus-Zeichen. Gehen Sie im neuen Fenster links auf **System Interpreter** und schließlich auf **Python 3.7**.

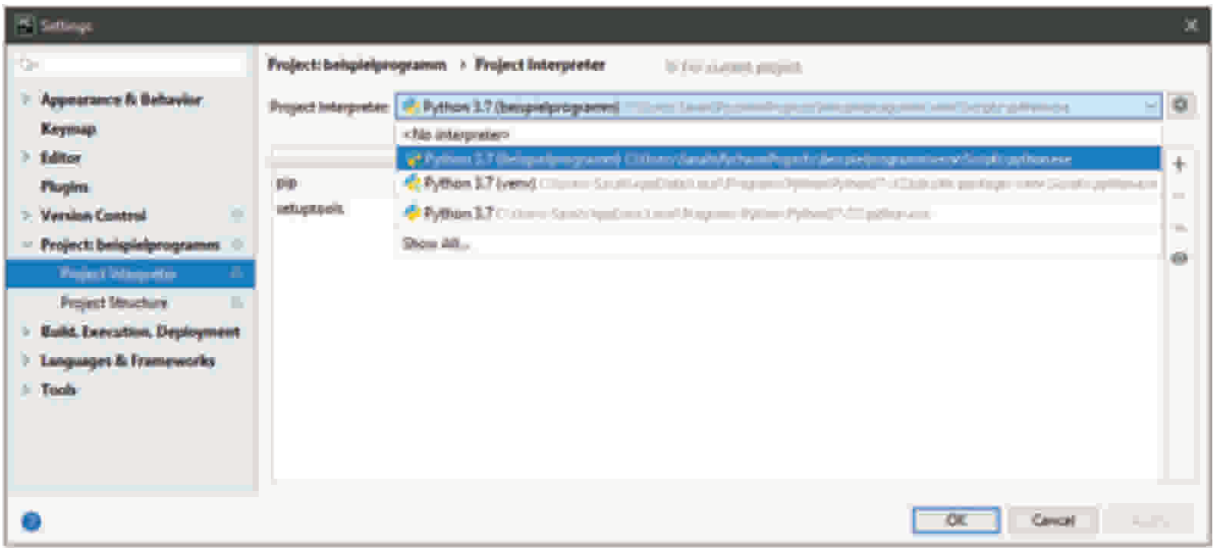

Abb. 2.10 Angabe des gewünschten Interpreters

Nun wird standardmäßig bei jedem neuen Projekt automatisch der zuvor installierte Interpreter verwendet. Sollten Sie für ein Projekt einen anderen Interpreter (beispielsweise Python 2.7) nutzen wollen, so können Sie dies in den Projekteinstellungen wieder ändern.

Über **File → Settings...** können Sie viele weitere Einstellungen vornehmen. Möchten Sie beispielsweise die Rechtschreibprüfung, die standardmäßig eingeschaltet ist, wieder deaktivieren, müssen Sie hierfür in der linken Auswahl unter **Editor → Inspections** den Haken rechts bei **Spelling → Typo** herausnehmen.

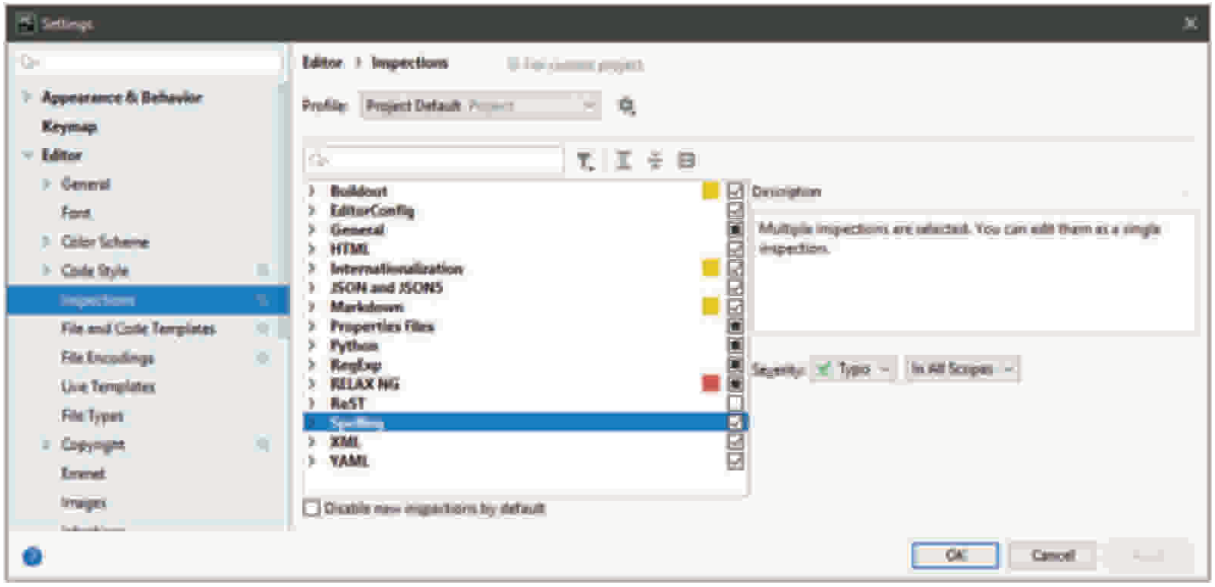

Abb. 2.11 Editor-Einstellungen ändern

Die Anzeige von Zeilennummern ist im Grunde eine sehr hilfreiche Option. Wenn Sie sie dennoch ausschalten möchten, können Sie sie unter **Editor → General → Ap-**

pearance → Show line numbers deaktivieren. Unter dem **Editor**-Eintrag lassen sich viele weitere Einstellungen, etwa zur Schrift und Farbdarstellung, anpassen.

Nun steht dem eigentlichen Programmieren nichts mehr im Wege. Bevor wir in die faszinierenden Möglichkeiten des Programmierens mit Python eintauchen und unsere ersten längeren Programme in der IDE schreiben, ausführen und überarbeiten lernen, werden wir im nächsten Kapitel einige grundlegende Befehle in einem Python-Prompt ausprobieren. Ab Kapitel 4 werden wir zur IDE wechseln und uns mit ihren unterschiedlichen Funktionen vertraut machen.