Python Crashkurs

Eine praktische, projektbasierte Programmiereinführung

2
Variablen und einfache Datentypen

In diesem Kapitel lernen Sie die verschiedenen Arten von Daten kennen, mit denen Sie in Python-Programmen arbeiten können. Außerdem erfahren Sie, wie Sie die Daten in Variablen speichern und diese Variablen in Ihren Programmen einsetzen.

Was bei der Ausführung von hello_world.py wirklich geschieht

Sehen wir uns nun genauer an, was Python macht, wenn Sie *hello_world.py* laufen lassen. Selbst bei einem so einfachen Programm erledigt Python eine ziemliche Menge an Arbeit:

hello_world.py

```
print("Hello Python world!")
```

Wenn Sie diesen Code ausführen, erhalten Sie folgende Ausgabe:

```
Hello Python world!
```

Die Dateiendung *.py* gibt an, dass es sich bei der Datei um ein Python-Programm handelt. Der Editor führt die Datei mithilfe des *Python-Interpreters* aus, der das Programm liest und dabei ermittelt, was jedes einzelne darin enthaltene Wort bedeutet. Sieht er beispielsweise das Wort `print`, gefolgt von Klammern, gibt er den Inhalt dieser Klammern auf dem Bildschirm aus.

Während Sie ein Programm schreiben, kennzeichnet der Editor die verschiedenen Teile des Programms durch unterschiedliche Farben. Beispielsweise erkennt er, dass es sich bei `print()` um den Namen einer Funktion handelt, und stellt das Wort in einer bestimmten Farbe dar. Er bemerkt auch, dass `"Hello Python world!"` kein Python-Code ist, und färbt diesen Satz daher anders ein. Diese sogenannte *Syntaxkennzeichnung* ist sehr praktisch, wenn Sie Ihre eigenen Programme schreiben.

Variablen

Versuchen wir nun, in *hello_world.py* eine Variable zu verwenden. Fügen Sie am Anfang der Datei eine neue Zeile ein und ändern Sie die zweite Zeile:

hello_world.py

```
message = "Hello Python world!"
print(message)
```

Wenn Sie dieses Programm ausführen, erhalten Sie die gleiche Ausgabe wie zuvor:

```
Hello Python world!
```

Was wir hinzugefügt haben, ist eine *Variable* namens `message`. Jede Variable enthält einen *Wert*, also eine Information, die mit ihr verknüpft ist. In diesem Fall ist der Wert der Text `"Hello Python world!"`.

Die Ergänzung um eine Variable macht dem Python-Interpreter etwas mehr Arbeit. Wenn er die erste Zeile verarbeitet, verknüpft er den Text `"Hello Python world!"` mit der Variablen `message`. In der zweiten Zeile angekommen, gibt er den Wert von `message` auf dem Bildschirm aus.

Wir wollen das Programm nun noch um eine zweite Meldung erweitern. Fügen Sie am Ende von *hello_world.py* eine leere Zeile und dann zwei neue Codezeilen ein:

```
message = "Hello Python world!"
print(message)
```

```
message = "Hello Python Crash Course world!"
print(message)
```

Wenn Sie jetzt *hello_world.py* ausführen, erhalten Sie eine Ausgabe von zwei Zeilen:

```
Hello Python world!
Hello Python Crash Course world!
```

Den Wert einer Variablen können Sie in einem Programm jederzeit ändern. Python merkt sich stets den aktuellen Wert.

Variablen benennen und verwenden

Wenn Sie in Python Variablen verwenden, müssen Sie einige Regeln und Richtlinien einhalten. Die Richtlinien dienen nur dazu, Code zu schreiben, der sich leichter lesen und verstehen lässt, doch wenn Sie die Regeln brechen, kann das einen Fehler zur Folge haben. Merken Sie sich daher folgende Punkte für den Umgang mit Variablen:

- Variablennamen dürfen nur Buchstaben, Ziffern und Unterstriche enthalten. Am Anfang darf ein Buchstabe oder ein Unterstrich stehen, aber keine Zahl. Beispielsweise können Sie eine Variable `message_1` nennen, aber nicht `1_message`.
- Leerzeichen sind in Variablennamen nicht erlaubt. Um Namensbestandteile zu trennen, können Sie Unterstriche verwenden. Beispielsweise ist `greeting_message` ein gültiger Variablenname, wohingegen `greeting message` einen Fehler hervorruft.
- Verwenden Sie keine Python-Schlüsselwörter und -Funktionsnamen als Variablennamen, also keine Wörter, die in Python für bestimmte Zwecke reserviert sind, z. B. das Wort `print` (siehe »Schlüsselwörter und integrierte Funktionen« in Anhang A).
- Variablennamen sollten kurz, aber aussagekräftig sein. Beispielsweise ist `name` besser als `n`, `student_name` besser als `s_n` und `name_length` besser als `length_of_persons_name`.
- Seien Sie bei der Verwendung des kleinen `l` und des großen `O` vorsichtig, da diese Buchstaben leicht mit den Zahlen `1` und `0` verwechselt werden können.

Es kann etwas Übung erfordern, sich gute Variablennamen auszudenken, vor allem, wenn Ihre Programme interessanter und komplizierter werden. Je mehr Programme Sie schreiben und je mehr Code anderer Leute Sie lesen, umso besser werden Sie jedoch darin, Variablen aussagekräftig zu benennen.

Hinweis

Momentan noch sollten Sie in Python ausschließlich Variablennamen in Kleinbuchstaben verwenden. Großbuchstaben rufen zwar keine Fehler hervor, haben aber eine besondere Bedeutung, die Sie in späteren Kapiteln noch kennenlernen werden.

Fehler bei Variablennamen vermeiden

Alle Programmierer machen Fehler, die meisten sogar täglich. Gute Programmierer wissen allerdings, wie sie diese Fehler ohne großen Aufwand beheben können. Als Beispiel wollen wir uns einen Fehler ansehen, wie er in der Anfangsphase häufig vorkommt, und zeigen, wie Sie ihn korrigieren können.

Zur Veranschaulichung schreiben wir Code, der absichtlich einen Fehler hervorruft. Geben Sie den folgenden Code ein – einschließlich des fett hervorgehobenen, falsch geschriebenen Wortes `mesage`:

```
message = "Hello Python Crash Course reader!"
print(mesage)
```

Wenn in einem Programm ein Fehler auftritt, versucht der Python-Interpreter sein Bestes, um Ihnen dabei zu helfen, die problematische Stelle zu finden. Dazu gibt er eine *Rückverfolgung* oder ein *Traceback* aus, wenn sich ein Programm nicht erfolgreich ausführen lässt. Dabei handelt es sich um eine Angabe der Stelle, an der der Interpreter auf Schwierigkeiten gestoßen ist. Bei unserem Beispielcode mit dem falsch geschriebenen Variablennamen gibt Python folgendes Traceback aus:

```
Traceback (most recent call last):
❶  File "hello_world.py", line 2, in <module>
❷    print(mesage)
       ^^^^^^
❸ NameError: name 'mesage' is not defined. Did you mean: 'message'?
```

Die Ausgabe bei ❶ zeigt, dass der Fehler in Zeile 2 der Datei *hello_world.py* aufgetreten ist. Außerdem zeigt der Interpreter die betreffende Zeile an, sodass wir den Fehler schneller erkennen können (❷), und er gibt an, um was für eine Art Fehler es sich handelt (❸). In diesem Fall ist ein *Namensfehler aufgetreten*, denn die auszugebende Variable `mesage` ist nicht definiert. Das heißt, Python kann den angegebenen Variablennamen nicht erkennen. Ein Namensfehler bedeutet entweder, dass wir vergessen haben, einen Wert für die Variable festzulegen, bevor wir sie verwenden, oder dass wir den Namen der Variablen falsch geschrieben haben. Wenn Python einen Variablennamen findet, der dem nicht erkannten ähnlich ist, fragt es, ob das der Name ist, den Sie verwenden wollten.

In diesem Fall ist der Fehler offensichtlich: Wir haben in der zweiten Zeile ein `s` im Variablennamen `message` vergessen. Der Python-Interpreter führt an unserem Code keine Rechtschreibprüfung durch, verlangt aber, dass die Variablennamen einheitlich geschrieben sind. Schauen Sie sich an, was passiert, wenn wir `message` beide Male falsch schreiben:

```
mesage = "Hello Python Crash Course reader!"
print(mesage)
```

In diesem Fall wird das Programm erfolgreich ausgeführt:

```
Hello Python Crash Course reader!
```

Da die Namen der Variablen übereinstimmen, sieht Python keine Probleme. Programmiersprachen sind streng, können aber nicht zwischen guter und schlechter Rechtschreibung unterscheiden. Daher kommt es bei Variablennamen und Code nicht auf die englischen oder deutschen Rechtschreib- und Grammatikregeln an.

Bei vielen Programmierfehlern handelt es sich um einfache Tippfehler in einer einzigen Programmzeile. Wenn Sie viel Zeit damit verbringen, solche Fehler aufzuspüren, lassen Sie sich gesagt sein, dass Sie sich in bester Gesellschaft befinden. Viele erfahrene und talentierte Programmierer sind Stunden damit beschäftigt, solche winzigen Fehler auszumerzen. Versuchen Sie, über solche Missgeschicke zu lachen und weiterzumachen. Sie werden Ihnen im Programmiererleben noch häufig unterlaufen.

Variablen sind Etiketten

Variablen werden oft mit Kisten verglichen, in denen Sie Werte ablegen können. Dieses Bild kann hilfreich sein, wenn Sie beginnen, Variablen zu verwenden, beschreibt aber nicht genau, wie Variablen intern in Python dargestellt werden. Es ist besser, sich Variablen als Etiketten vorzustellen, denen Sie Werte zuweisen. Sie können auch sagen, dass eine Variable auf einen bestimmten Wert verweist.

Bei den ersten Programmen, die Sie schreiben, wird dieser Unterschied wahrscheinlich nicht viel ausmachen, aber je früher Sie sich darüber im Klaren sind, umso besser. Mit Sicherheit werden Sie irgendwann einmal mit dem überraschenden Verhalten einer Variablen konfrontiert. Ein genaues Verständnis der tatsächlichen Funktionsweise von Variablen kann Ihnen dann helfen zu erkennen, was in Ihrem Code vor sich geht.

Hinweis

Um neue Programmierprinzipien zu verstehen, ist es am besten, sie in eigenen Programmen anzuwenden. Wenn Sie bei einer Aufgabe in diesem Buch nicht weiterkommen, versuchen Sie, zunächst einmal etwas anderes zu tun. Wenn Sie immer noch nicht weiterkommen, schauen Sie sich noch einmal die entsprechenden Abschnitte in dem Kapitel an. Wenn Sie Hilfe brauchen, können Sie in Anhang C erfahren, an wen Sie sich wenden können.

Probieren Sie es selbst aus!

Schreiben Sie für jede einzelne Übung ein eigenes Programm und speichern Sie diese Programme jeweils mit einem Dateinamen, der der Python-Konvention genügt, der also nur aus Kleinbuchstaben und Unterstrichen besteht, z. B. *simple_message.py* und *simple_messages.py*.

2-1 Einfache Nachricht: Weisen Sie eine Nachricht einer Variablen zu und geben Sie sie aus.

2-2 Einfache Nachrichten: Weisen Sie eine Nachricht einer Variablen zu und geben Sie sie aus. Ändern Sie dann den Wert der Variablen in eine neue Nachricht und geben Sie diese ebenfalls aus.

Strings

Die meisten Programme definieren und erfassen Daten einer bestimmten Art und machen dann irgendetwas Sinnvolles damit. Daher ist es hilfreich, die verschiedenen Arten von Daten in Kategorien einzuteilen. Der erste Datentyp, den wir uns dabei ansehen, ist der String. Strings erscheinen auf den ersten Blick ganz einfach, lassen sich aber sehr vielseitig verwenden.

Bei einem *String* handelt es sich um eine Folge von Zeichen. In Python wird alles, was in Anführungszeichen eingeschlossen ist, als String aufgefasst. Dabei können Sie sowohl einfache als auch doppelte Anführungszeichen verwenden:

```
"Dies ist ein String."
'Dies ist auch ein String.'
```

Das ermöglicht es auch, innerhalb von Strings Anführungszeichen und Apostrophe zu verwenden:

```
'I told my friend, "Python is my favorite language!"'
"The language 'Python' is named after Monty Python, not the snake."
"One of Python's strengths is its diverse and supportive community."
```

Sehen wir uns noch weitere Möglichkeiten zur Verwendung von Strings an.

Groß- und Kleinschreibung mithilfe von Methoden ändern

Eine der einfachsten Aufgaben bei der Arbeit mit Strings besteht darin, die Groß- und Kleinschreibung zu ändern. Schauen Sie sich den folgenden Code an und versuchen Sie herauszufinden, was passiert:

name.py

```
name = "ada lovelace"
print(name.title())
```

Speichern Sie die Datei als *name.py* und führen Sie sie aus. Dabei erhalten Sie folgende Ausgabe:

```
Ada Lovelace
```

In diesem Beispiel verweist die Variable `name` auf den kleingeschriebenen String `"ada lovelace"`. Im Aufruf von `print()` steht aber hinter der Variablen noch die Methode `title()`. Eine *Methode* ist eine Aktion, die Python an Daten ausführen kann. Der Punkt in `name.title()` weist Python an, die Methode `title()` auf die Variable `name` anzuwenden. Hinter einem Methodennamen steht immer ein Klammernpaar, da viele Methoden zusätzliche Angaben benötigen, um ihre Aufgaben ausführen zu können. Diese Angaben werden dann in die Klammern geschrieben. Die Funktion `title()` braucht jedoch keine zusätzlichen Angaben, weshalb die Klammern hier leer sind.

Die Methode `title()` zeigt alle Wörter mit großem Anfangsbuchstaben an. Das ist praktisch, wenn Ihr Programm etwa die Eingabewerte `Ada`, `ADA` und `ada` alle als denselben Namen erkennen und stets als `Ada` ausgeben soll.

Es gibt noch mehrere andere praktische Methoden, um die Groß- und Kleinschreibung zu ändern. Beispielsweise können Sie einen String wie folgt komplett in Groß- oder Kleinbuchstaben umwandeln:

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

Dies führt zu folgender Ausgabe:

```
ADA LOVELACE
ada lovelace
```

Die Methode `lower()` ist insbesondere für die Speicherung von Daten nützlich. Oft können Sie nicht darauf vertrauen, dass die Benutzer Ihres Programms die Groß- und Kleinschreibung richtig machen, weshalb Sie alle eingegebenen Strings vor der Speicherung in Kleinbuchstaben umwandeln. Wenn Sie die Informationen

anzeigen wollen, können Sie dann die Art von Groß- und Kleinschreibung verwenden, die für den jeweiligen String geeignet ist.

Variablen in Strings verwenden

Es kann vorkommen, dass Sie innerhalb eines Strings den Wert einer Variablen verwenden möchten, etwa wenn Sie zwei Variablen für den Vor- und den Nachnamen einer Person haben und zur Anzeige des vollständigen Namens kombinieren möchten:

full_name.py

```
first_name = "ada"
last_name = "lovelace"
❶ full_name = f"{first_name} {last_name}"
print(full_name)
```

Um den Wert einer Variablen in einen String einzufügen, stellen Sie den Buchstaben `f` unmittelbar vor das öffnende Anführungszeichen (❶). Die Namen von Variablen, die Sie innerhalb eines Strings verwenden wollen, müssen Sie in geschweifte Klammern einschließen. Bei der Anzeige des Strings ersetzt Python die Variablen durch ihren Wert.

Diese Strings werden als *F-Strings* bezeichnet. Das *F* steht für »Formatierung«, da Python die Strings formatiert, indem es die von geschweiften Klammern umgebenen Variablennamen durch die entsprechenden Werte ersetzt. Die Ausgabe des vorstehenden Codes sieht daher wie folgt aus:

```
ada lovelace
```

Mit F-Strings lässt sich eine Menge anfangen. Beispielsweise können Sie damit vollständige Benachrichtigungen aus den mit Variablen verknüpften Informationen zusammenstellen:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
❶ print(f"Hello, {full_name.title()}!")
```

Hier wird der vollständige Name in einem Satz zur Begrüßung der Benutzerin verwendet (❶), wobei die Groß- und Kleinschreibung mit der Methode `title()` korrigiert wird. Dieser Code gibt die folgende einfache, aber wohlformatierte Begrüßung aus:

```
Hello, Ada Lovelace!
```

Sie können mithilfe von F-Strings auch eine längere Nachricht zusammenbauen und diese dann einer Variablen zuweisen:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
❶ message = f"Hello, {full_name.title()}!"
❷ print(message)
```

Dieser Code gibt ebenfalls die Meldung `"Hello, Ada Lovelace!"` aus, weist sie aber auch einer Variablen zu (❶), was den Aufruf von `print()` bei ❷ viel einfacher macht.

Weißraum hinzufügen

Alle nicht druckbaren Zeichen wie Leerzeichen, Tabulatoren und Zeilenumbrüche werden als *Weißraum* bezeichnet. Damit können Sie die Ausgabe so gliedern, dass sie besser lesbar ist.

Um einen Tabulator hinzuzufügen, verwenden Sie wie bei ❶ die Zeichenfolge `\t`:

```
>>> print("Python")
Python
❶ >>> print("\tPython")
    Python
```

Wollen Sie in einem String einen Zeilenumbruch vornehmen, verwenden Sie die Zeichenfolge `\n`:

```
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript
```

Sie können Tabulatoren und Zeilenumbrüche auch kombinieren. Die Zeichenfolge `\n\t` weist Python an, in die nächste Zeile zu wechseln und diese mit einem Tabulator zu beginnen:

```
>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")
Languages:
    Python
    C
    JavaScript
```

Zeilenumbrüche und Tabulatoren werden Sie in den nächsten beiden Kapiteln besonders zu schätzen lernen, in denen Sie damit beginnen werden, aus wenigen Codezeilen viele Ausgabezeilen zu produzieren.

Weißraum entfernen

Zusätzlicher Weißraum in Programmen kann irreführend sein. Für Sie als Programmierer mögen `'python'` und `'python '` zwar sehr ähnlich aussehen, aber für ein Programm sind das zwei völlig verschiedene Strings. Python erkennt das zusätzliche Leerzeichen in `'python '` und misst ihm eine Bedeutung bei – sofern Sie nicht ausdrücklich sagen, dass es bedeutungslos ist.

Es ist wichtig, den Weißraum im Auge zu behalten, vor allem, wenn Sie zwei Strings vergleichen, etwa bei der Überprüfung von Benutzernamen bei der Anmeldung an einer Website. Zusätzlicher Weißraum kann aber auch in einfacheren Situationen zu Verwirrung führen. Daher ist es gut, dass Python einfache Möglichkeiten bietet, um überflüssigen Weißraum von Benutzereingaben zu entfernen.

Python ist in der Lage, am rechten und am linken Ende eines Strings nach Weißraum zu suchen. Um jeglichen Weißraum am rechten Ende zu entfernen, verwenden Sie die Methode `rstrip()`.

```
❶ >>> favorite_language = 'python '
❷ >>> favorite_language
  'python '
❸ >>> favorite_language.rstrip()
  'python'
❹ >>> favorite_language
  'python '
```

Der bei ❶ zu `favorite_language` zugewiesene Wert enthält zusätzlichen Weißraum am Ende des Strings. Wenn Sie Python in einer Terminalsitzung nach diesem Wert fragen, können Sie das Leerzeichen am Ende erkennen (❷). Durch die Anwendung der Methode `rstrip()` auf `favorite_language` bei ❸ wird dieses Leerzeichen entfernt – allerdings nur vorübergehend. Wenn Sie den Wert von `favorite_language` erneut abrufen, stellen Sie fest, dass der String immer noch genauso aussieht, wie er gespeichert wurde, nämlich mit dem zusätzlichen Weißraum (❹).

Um den Weißraum endgültig zu entfernen, müssen Sie den bereinigten Wert der Variablen zuweisen:

```
  >>> favorite_language = 'python '
❶ >>> favorite_language = favorite_language.rstrip()
  >>> favorite_language
  'python'
```

Hier entfernen Sie den Weißraum vom rechten Ende des Strings und weisen den resultierenden Wert dann der ursprünglichen Variablen zu, wie bei (❶) gezeigt. Beim Programmieren werden Sie oft einen Variablenwert ändern und anschließend den neuen Wert der Variablen zuweisen. Dadurch können Sie einen Variablenwert im Rahmen der Programmausführung oder in Reaktion auf eine Benutzereingabe ändern.

Natürlich können Sie Weißraum auch vom linken Ende eines Strings oder von beiden Enden zugleich entfernen. Dazu verwenden Sie die Methode `lstrip()` bzw. `strip()`:

```
❶ >>> favorite_language = ' python '
❷ >>> favorite_language.rstrip()
  ' python'
❸ >>> favorite_language.lstrip()
  'python '
❹ >>> favorite_language.strip()
  'python'
```

In diesem Beispiel verwenden wir einen Wert mit Weißraum am Anfang und am Ende (❶). Dann entfernen wir den Weißraum von der rechten Seite (❷), von der linken Seite (❸) und von beiden Seiten (❹). Spielen Sie ruhig ein bisschen mit diesen Kürzungsfunktionen herum, um sich mit der Stringbearbeitung vertraut zu machen. In der Praxis werden sie vor allem dazu verwendet, Benutzereingaben zu bereinigen, bevor sie in einem Programm gespeichert werden.

Entfernen von Präfixen

Bei der Arbeit mit Strings besteht eine weitere häufige Aufgabe darin, ein Präfix zu entfernen. Nehmen wir eine URL mit dem üblichen Präfix *https://*. Wir möchten dieses Präfix entfernen, damit wir uns auf den Teil der URL konzentrieren können, den die Benutzer in eine Adressleiste eingeben müssen. So wirds gemacht:

```
>>> nostarch_url = 'https://nostarch.com'
>>> nostarch_url.removeprefix('https://')
'nostarch.com
```

Geben Sie den Namen der Variablen ein, gefolgt von einem Punkt, und dann die Methode `removeprefix()`. Innerhalb der Klammern geben Sie das Präfix an, das Sie aus der ursprünglichen Zeichenkette entfernen möchten. Wie die Methoden zum Entfernen von Leerzeichen lässt `removeprefix()` die ursprüngliche Zeichenkette unverändert. Wenn Sie den neuen Wert mit dem entfernten Präfix behalten wollen, weisen Sie ihn entweder der ursprünglichen Variable erneut zu oder weisen Sie ihn einer neuen Variablen zu.

```
>>> simple_url = nostarch_url.removeprefix('https://')
```

Wenn Sie eine URL in einer Adressleiste sehen und der Teil *https://* nicht angezeigt wird, verwendet der Browser wahrscheinlich im Hintergrund eine Methode wie `removeprefix()`.

Syntaxfehler bei der Stringverarbeitung vermeiden

Hin und wieder werden Sie einem *Syntaxfehler* begegnen. Dabei kann Python einen Teil des Programms nicht als gültigen Python-Code erkennen. Wenn Sie beispielsweise innerhalb von einfachen Anführungszeichen einen Apostroph verwenden, interpretiert Python alles, was zwischen dem öffnenden Anführungszeichen und dem Apostroph steht, als String und glaubt, dass es sich bei dem Rest um Python-Code handelt, was natürlich zu einem Fehler führt.

Das folgende Beispiel zeigt, wie Sie einfache und doppelte Anführungszeichen korrekt verwenden. Speichern Sie das Programm als *apostrophe.py* und führen Sie es aus:

apostrophe.py

```
message = "One of Python's strengths is its diverse community."
print(message)
```

Da der Apostroph hier innerhalb eines von doppelten Anführungszeichen begrenzten Strings steht, kann der Python-Interpreter den String korrekt lesen:

```
One of Python's strengths is its diverse community.
```

Wenn Sie dagegen einfache Anführungszeichen verwenden, kann Python nicht erkennen, wo der String wirklich enden soll:

```
message = 'One of Python's strengths is its diverse community.'
print(message)
```

In diesem Fall erhalten Sie folgende Ausgabe:

```
  File "apostrophe.py", line 1
    message = 'One of Python's strengths is its diverse community.'
                                                                  ^ ❶
SyntaxError: unterminated string literal (detected at line 1)
```

Diese Ausgabe zeigt, dass bei ❶ unmittelbar hinter dem letzten Anführungszeichen (dem Apostroph) ein Syntaxfehler aufgetreten ist, da der Interpreter einen Teil des Programms nicht als gültigen Python-Code erkannt hat und vermutet, dass das mit einem falsch benannten String zu tun hat. Es gibt verschiedene Ursachen für

Fehler, und ich werde im Laufe unserer Erörterung jeweils auf übliche Fehlerquellen hinweisen. Während Sie noch lernen, Python-Code zu schreiben, werden Sie häufig Syntaxfehlern begegnen. Die Fehlermeldungen dieser Art sind ziemlich allgemein gehalten, sodass es schwierig und frustrierend sein kann, die Fehler aufzuspüren und zu beheben. Wenn Sie bei einem besonders hartnäckigen Fehler nicht weiterkommen, schlagen Sie in Anhang C nach.

Hinweis

Die Syntaxkennzeichnung in Ihrem Editor kann Ihnen helfen, einige Syntaxfehler schneller zu erkennen. Wenn Code in der Farbe für Text oder Text in der Farbe für Code angezeigt wird, dann haben Sie wahrscheinlich irgendwo in der Datei ein Anführungszeichen nicht richtig gesetzt.

Probieren Sie es selbst aus!

Speichern Sie für die folgenden Übungen die einzelnen Programme jeweils in eigenen Dateien mit Namen wie *name_cases.py*. Wenn Sie nicht mehr weiterkommen, legen Sie eine Pause ein oder sehen Sie sich die Vorschläge in Anhang C an.

2-3 Persönliche Nachricht: Weisen Sie den Namen einer Person einer Variablen zu und geben Sie eine einfache Nachricht an diese Person aus, z. B.: »Hello Eric, would you like to learn some Python today?«

2-4 Groß- und Kleinschreibung von Namen: Weisen Sie den Namen einer Person einer Variablen zu und geben Sie den Namen in Kleinbuchstaben, in Großbuchstaben sowie mit großen Anfangsbuchstaben aus.

2-5 Bekanntes Zitat: Geben Sie ein bekanntes Zitat und den Namen von dessen Urheber aus. Die Ausgabe sollte so aussehen wie im folgenden Beispiel, einschließlich der Anführungszeichen:

```
Albert Einstein once said, "A person who never made a mistake never tried
anything new."
```

2-6 Bekanntes Zitat 2: Wiederholen Sie Übung 2-5, weisen Sie aber diesmal den Namen des Urhebers der Variablen `famous_person` zu. Bauen Sie die Nachricht zusammen, weisen Sie sie der neuen Variablen `message` zu und geben Sie sie aus.

2-7 Namen bereinigen: Weisen Sie den Namen einer Person einschließlich einiger Weißraumzeichen am Anfang und Ende einer Variablen zu. Verwenden Sie dabei sowohl `\t` als auch `\n` jeweils mindestens einmal.

Geben Sie den Namen einmal einschließlich Weißraum aus und dann dreimal mithilfe der Funktionen `lstrip()`, `rstrip()` und `strip()`.

→

2-8 Dateierweiterungen: Python verfügt über eine Methode `removesuffix()`, die genau wie `removeprefix()` funktioniert. Weisen Sie den Wert `'python_notes.txt'` einer Variablen namens `filename` zu. Verwenden Sie dann die Methode `removesuffix()`, um den Dateinamen ohne die Dateierweiterung anzuzeigen, wie es einige Dateibrowser tun.

Zahlen

Zahlen werden in der Programmierung sehr häufig verwendet, etwa um Spielstände darzustellen, um Daten zu visualisieren, um Informationen in Webanwendungen zu speichern usw. Je nachdem, wie die Zahlen verwendet werden, behandelt Python sie auf unterschiedliche Weise. Als Erstes sehen wir uns an, wie Python mit ganzen Zahlen (*Integer*) umgeht, da die Arbeit mit ihnen am einfachsten ist.

Integer

In Python können Sie Integer addieren (+), subtrahieren (-), multiplizieren (*) und dividieren (/).

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
```

In einer Terminalsitzung gibt Python einfach das Ergebnis der Operation zurück. Zur Darstellung von Exponenten verwendet Python zwei Sternchen:

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

Python richtet sich auch nach der mathematischen Reihenfolge von Operationen, sodass Sie mehrere Operationen in einem Ausdruck kombinieren können. Um die Reihenfolge zu ändern, können Sie Klammern angeben:

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
```

Die Leerzeichen in diesen Beispielen haben keine Auswirkungen darauf, wie Python die Ausdrücke auswertet. Sie ermöglichen es Ihnen nur, beim Lesen schneller zu erkennen, welche Operationen Priorität haben.

Fließkommazahlen

Zahlen mit einem Dezimalpunkt werden in den meisten Programmiersprachen und auch in Python als *Fließkommazahlen* bezeichnet, weil der Dezimalpunkt (der dem Komma in der deutschen Schreibweise entspricht) an beliebiger Stelle stehen kann. Alle Programmiersprachen müssen in der Lage sein, Dezimalzahlen korrekt zu handhaben, unabhängig davon, wo der Dezimalpunkt steht.

Meistens können Sie Dezimalzahlen verwenden, ohne sich Gedanken über das Verhalten zu machen. Geben Sie einfach die gewünschten Zahlen ein, und Python wird sehr wahrscheinlich das tun, was Sie erwarten:

```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
```

Es kann jedoch auch vorkommen, dass das Ergebnis eine willkürliche Anzahl von Stellen aufweist:

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

Das kann in allen Programmiersprachen geschehen und ist kein Grund zur Besorgnis. Python versucht, das Ergebnis so genau wie möglich wiederzugeben, was aufgrund der Art und Weise, wie die Zahlen auf dem Computer intern dargestellt werden, zu Schwierigkeiten führt. Sie können solche überzähligen Stellen vorläufig ignorieren. Wie Sie bei Bedarf damit umgehen, erfahren Sie in den Projekten in Teil II.

Integer und Fließkommazahlen

Wenn Sie zwei Zahlen dividieren, erhalten Sie immer eine Fließkommazahl, auch wenn es sich bei den beiden Zahlen um Integer handelt und das Ergebnis eine ganze Zahl ist:

```
>>> 4/2
2.0
```

Ebenso erhalten Sie eine Fließkommazahl, wenn Sie in den anderen Operationen Integer und Fließkommazahlen mischen:

```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
```

In allen Operationen, an denen Fließkommazahlen beteiligt sind, verwendet Python standardmäßig Fließkommazahlen, auch wenn das Ergebnis eine ganze Zahl ist.

Unterstriche in Zahlen

Wenn Sie sehr große Zahlen schreiben, können Sie der besseren Lesbarkeit halber Unterstriche als Tausendertrennzeichen verwenden:

```
>>> universe_age = 14_000_000_000
```

Bei der Ausgabe zeigt Python aber nur die Ziffern einer solchen mit Unterstrichen definierten Zahl an:

```
>>> print(universe_age)
14000000000
```

Beim Speichern solcher Werte ignoriert Python die Unterstriche. Auch wenn Sie eine Zahl mit den Unterstrichen auf andere Weise aufteilen als in Dreiergruppen, bleibt der Wert unbeeinträchtigt. Für Python sind `1000`, `1_000` und `10_00` identisch. Diese Verwendung von Unterstrichen ist sowohl für Integer als auch für Fließkommazahlen möglich.

Mehrfachzuweisung

Sie können innerhalb einer Zeile Code Werte zu mehr als einer Variablen zuweisen. Das macht die Programme kürzer und leichter lesbar. Diese Technik wird vor allem bei der Initialisierung eines zusammengehörigen Satzes von Variablen angewendet.

Beispielsweise können Sie die Variablen `x`, `y` und `z` wie folgt mit 0 initialisieren:

```
>>> x, y, z = 0, 0, 0
```

Die Variablennamen und die Werte müssen Sie jeweils durch Kommata trennen. Python weist dann jeder Variablen den entsprechenden Wert in der Reihenfolge zu. Diese Zuordnung erfolgt korrekt, sofern es gleich viele Variablen wie Werte gibt.

Konstanten

Eine *Konstante* ist eine Variable, deren Wert während der gesamten Lebensdauer des Programms unverändert bleibt. Python hat zwar keinen vordefinierten Typ für Konstanten, allerdings verwenden Python-Programmierer vereinbarungsgemäß Großbuchstaben, um anzuzeigen, dass eine Variable als Konstante behandelt und nie geändert werden soll:

```
MAX_CONNECTIONS = 5000
```

Wenn Sie eine Variable in Ihrem Code als Konstante behandeln wollen, sollten Sie ihren Namen ebenfalls in Großbuchstaben schreiben.

Probieren Sie es selbst aus!

2-9 Alles auf 8: Schreiben Sie eine Addition, eine Subtraktion, eine Multiplikation und eine Division, die jeweils 8 ergeben. Schließen Sie die Operationen in `print`-Anweisungen ein, damit Sie das Ergebnis sehen können. Ihr Programm sollte aus vier Zeilen wie den folgenden bestehen:

```
print(5+3)
```

Die Ausgabe soll vier Zeilen umfassen, in denen jeweils das Ergebnis 8 auf der rechten Seite steht.

2-10 Lieblingszahl: Weisen Sie Ihre Lieblingszahl einer Variablen zu. Stellen Sie mithilfe dieser Variablen eine Nachricht zusammen, in der diese Lieblingszahl angegeben wird, und geben Sie diese Nachricht aus.

Kommentare

In allen Programmiersprachen gibt es die äußerst nützliche Einrichtung von Kommentaren. Alles, was Sie bis jetzt in den Beispielprogrammen geschrieben haben, war Python-Code. Wenn Ihre Programme länger und komplizierter werden, sollten Sie jedoch auch Hinweise hinzufügen, die Ihre Vorgehensweise zur Lösung des Problems beschreiben. In einem *Kommentar* können Sie solche Hinweise in normaler Alltagssprache in Ihre Programme aufnehmen.

Wie werden Kommentare geschrieben?

In Python dient das amerikanische Nummernsymbol (#) zur Kennzeichnung eines Kommentars. Alles, was in einer Zeile auf dieses Zeichen folgt, wird vom Python-Interpreter ignoriert, wie das folgende Beispiel zeigt:

comment.py

```
# Begrüßt alle mit "Hello".
print("Hello Python people!")
```

Hier ignoriert Python die erste Zeile und führt nur die zweite aus:

```
Hello Python people!
```

Was für Kommentare sind sinnvoll?

Der Hauptzweck von Kommentaren besteht darin, zu erklären, was der Code tun soll und wie Sie versuchen, dies zu erreichen. Während Sie an einem Projekt arbeiten, ist Ihnen völlig klar, wie die einzelnen Teile zusammenwirken, aber wenn Sie sich nach einiger Zeit wieder mit einem älteren Projekt beschäftigen, haben Sie wahrscheinlich einige der Einzelheiten vergessen. Sie könnten den Code zwar ausgiebig studieren und herauszufinden versuchen, wie die Einzelteile zusammenwirken, aber wenn Sie Ihre Vorgehensweise in Form von guten, verständlichen Kommentaren erklärt haben, hilft Ihnen das, viel Zeit zu sparen.

Wenn Sie Programmierung als Beruf anstreben oder mit anderen Programmierern zusammenarbeiten wollen, müssen Sie aussagekräftige Kommentare schreiben. Heute wird die meiste Software im Team geschrieben. Das kann eine Gruppe von Angestellten einer Firma sein, aber auch eine Gruppe von Personen, die gemeinsam an einem Open-Source-Projekt arbeiten. Erfahrene Programmierer erwarten, Kommentare im Code zu sehen, weshalb Sie es sich am besten gleich angewöhnen sollten, beschreibende Kommentare in Ihre Programme aufzunehmen. Klare und prägnante Kommentare zu schreiben gehört zu den nützlichsten Dingen, die Sie sich als angehender Programmierer angewöhnen können.

Wenn Sie an einer Stelle überlegen, ob Sie einen Kommentar einfügen sollten oder nicht, fragen Sie sich, ob Sie mehrere verschiedene Ansätze erwogen haben, bevor Sie auf eine sinnvolle Vorgehensweise gestoßen sind. Wenn das der Fall ist, dann beschreiben Sie Ihre Lösung in einem Kommentar. Es ist viel einfacher, später überflüssige Kommentare zu löschen, als ein unzureichend kommentiertes Programm nachträglich mit Kommentaren zu versehen. Von jetzt an werde ich auch in den Beispielen in diesem Buch Kommentare angeben, um einzelne Abschnitte des Codes zu erklären.

Probieren Sie es selbst aus!

2-11 Kommentare hinzufügen: Wählen Sie zwei Ihrer bisher geschriebenen Programme aus und fügen Sie in jedes mindestens einen Kommentar ein. Wenn Sie nicht wissen, was Sie schreiben sollen, da die bisherigen Programme extrem einfach waren, fügen Sie einfach am Anfang der Programmdatei Ihren Namen und das Datum ein und geben Sie in einem Satz an, was das Programm macht.

The Zen of Python

Erfahrene Python-Programmierer ermuntern Sie dazu, Komplexität zu vermeiden und nach Möglichkeit Einfachheit anzustreben. Die Grundhaltung der Python-Community kommt in dem Text »The Zen of Python« von Tim Peters zum Ausdruck. Diese knappe Aufstellung von Prinzipien zum Schreiben von gutem Python-Code können Sie sich ansehen, indem Sie im Interpreter `import this` eingeben.

```
>>> import this
The Zen of Python, by Tim Peters
```

Ich werde »The Zen of Python« hier nicht komplett wiedergeben, sondern nur einige Zeilen daraus anführen, um Ihnen zu zeigen, warum diese Prinzipien für Sie als angehender Python-Programmierer so wichtig sind.

```
Beautiful is better than ugly.
```

Für Python-Programmierer kann Code schön und elegant sein. Bei der Programmierung geht es darum, Probleme zu lösen. Programmierer haben schon immer gut gestaltete, effiziente und auch schöne Lösungen für Probleme gewürdigt. Wenn Sie mehr über Python lernen und mehr Code schreiben, wird Ihnen eines Tages jemand über die Schulter schauen und sagen: »Wow, das ist wirklich schöner Code!«

```
Einfach ist besser als komplex.
```

Wenn Sie die Wahl zwischen einer einfachen und einer komplexen Lösung haben, die beide funktionieren, dann entscheiden Sie sich für die einfache. Ihr Code lässt sich dann einfacher pflegen, und Sie und andere können später einfacher darauf aufbauen.

```
Komplex ist besser als kompliziert.
```

Das Leben ist nicht immer einfach, und manchmal gibt es keine einfache Lösung für ein Problem. Nehmen Sie in einem solchen Fall die einfachste funktionierende Lösung.

```
Lesbarkeit zählt.
```

Selbst wenn Ihr Code komplex ist, sollten Sie stets darauf achten, ihn lesbar zu gestalten. Nehmen Sie informative Kommentare in den Code auf.

```
Es sollte eine -- und zwar vorzugsweise nur eine -- offensichtliche Möglich-
keit geben.
```

Wenn zwei Python-Programmierer gebeten werden, dasselbe Problem zu lösen, sollten dabei ziemlich ähnliche Lösungen herauskommen. Das heißt jedoch nicht, dass es in der Programmierung keinen Platz für Kreativität gäbe. Ganz im Gegenteil! Aber Programmierung besteht zum großen Teil aus kleinen, gängigen Lösungsansätzen für einfache Lösungen im Rahmen größerer Projekte, die mehr Kreativität erfordern. Die grundlegenden Mechanismen Ihrer Programme sollten auch für andere Python-Programmierer sofort einsichtig sein.

```
Jetzt ist besser als nie.
```

Sie können Ihr ganzes Leben damit zubringen, alle Feinheiten von Python und der Programmierung im Allgemeinen zu lernen, aber dann würden Sie niemals dazu kommen, ein Projekt fertigzustellen. Versuchen Sie nicht, perfekten Code zu schreiben, sondern schreiben Sie Code, der funktioniert. Dann können Sie immer noch entscheiden, ob Sie den Code verbessern oder mit einem neuen Projekt anfangen wollen.

Behalten Sie diese Prinzipien von Einfachheit und Klarheit im Hinterkopf, wenn Sie die nächsten Kapitel durcharbeiten und in die anspruchsvolleren Themen einsteigen. Erfahrene Programmierer werden Ihren Code dann stärker würdigen, Ihnen Rückmeldung geben und gern an interessanten Projekten mit Ihnen zusammenarbeiten.

Probieren Sie es selbst aus!

2-12 The Zen of Python: Geben Sie in einer Python-Terminalsitzung `import this` ein und schauen Sie sich auch die anderen aufgeführten Prinzipien an.

Zusammenfassung

In diesem Kapitel haben Sie gelernt, wie Sie mit Variablen arbeiten. Sie haben erfahren, dass Sie beschreibende Variablennamen verwenden sollten, und es wurde Ihnen gezeigt, wie Sie Namens- und Syntaxfehler beheben können. Außerdem ging es darum, was Strings sind und wie Sie sie in Kleinbuchstaben, in Großbuchstaben und mit großem Anfangsbuchstaben anzeigen können. Des Weiteren haben Sie gelernt, wie Sie unnötige Elemente aus verschiedenen Teilen eines Strings entfernen, um Ausgaben sauber zu formatieren. Sie haben auch schon Integer und Fließkommazahlen verwendet und einige Möglichkeiten für die Arbeit mit numerischen Daten kennengelernt. Zudem haben Sie erfahren, wie Sie erklärende Kommentare schreiben, um Ihren Code für sich selbst und andere leichter verständlich zu machen. Am Ende des Kapitels wurde Ihnen das Grundprinzip vorgestellt, Ihren Code so einfach wie möglich zu halten.

In Kapitel 3 erfahren Sie, wie Sie Gruppen von Informationen in besonderen Datenstrukturen, sogenannten *Listen,* speichern können und wie Sie eine solche Liste durcharbeiten und die darin enthaltenen Informationen bearbeiten.