Inhalt

5.4 Dynamische Typisierung: viele Freiheiten bei der Nutzung von Variablen

Wie bereits erwähnt wurde, ist es bei Python nicht notwendig, den Typ einer Variablen anzugeben. Darüber hinaus zeichnet sich Python durch eine dynamische Typisierung aus. Bei den meisten Programmiersprachen wird der Typ einer Variablen bereits zu Beginn festgelegt. Danach ist es nicht mehr möglich, ihn zu verändern. Wenn man eine Variable mit einem anderen Typ benötigt, ist es notwendig, einen neuen Namen zu verwenden. Bei der dynamischen Typisierung wird der Variablentyp und damit der benötigte Speicherplatz hingegen erst bei der Ausführung festgelegt. Das macht es möglich, ihn innerhalb des Programms zu ändern.

Computerprogramme unterscheiden beispielsweise zwischen ganzen Zahlen und Fließkommazahlen. Der Grund dafür liegt darin, dass der hierfür benötigte Speicherplatz sehr unterschiedlich ist. Bei Programmiersprachen mit statischer Typisierung ist es nicht möglich, einer Variablen, die einmal als ganze Zahl festgelegt wurde, Nachkommastellen zuzuweisen. In Python stellt dies jedoch kein Problem dar:

```
a = 5
print ("Wert der Variablen a: ",a)
a = 5.3
print ("Wert der Variablen a: ",a)
```

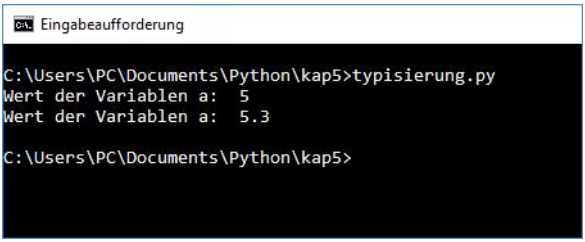

Abbildung 19: Die Variable verändert ihren Typ

Nun könnte man einwenden, dass es sich bereits zu Beginn um eine Fließkommazahl handeln könnte – schließlich ist es nicht notwendig, hierbei eine Nachkommastelle anzugeben, wenn diese Null beträgt. Dass es dennoch einen Unterschied gibt, zeigt folgendes Beispiel:

```
a = 5
print ("Wert der Variablen a: ",a)
a = a + 0.0
print ("Wert der Variablen a: ",a)
```

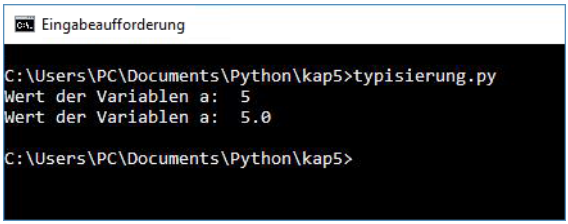

Abbildung 20: Die Ausgabe als Ganzzahl und als Fließkommazahl

Dieses Programm addiert 0,0 zum Wert der Variablen a. Aus mathematischer Sicht wird ihr Wert dabei nicht verändert. Dennoch kommt es zu einem Unterschied bei der Ausgabe: Nach dieser Operation wird die Nachkommastelle angegeben – selbst wenn diese Null beträgt. Der Grund dafür liegt darin, dass das Ergebnis einer Addition, an der eine Fließkommazahl beteiligt ist, stets ebenfalls eine Fließkommazahl ist. Dieser Regel folgend wandelt Python den Typ der Variablen a um, obwohl die mathematische Operation bedeutungslos ist. Das bewirkt den Unterschied in der Ausgabe. Dieses Beispiel zeigt außerdem, dass Python die Änderung ganz automatisch vornimmt, wenn die entsprechenden Rechenoperationen dies erforderlich machen.

Dabei ist es nicht nur erlaubt, ganze Zahlen in Fließkommazahlen umzuwandeln. Es ist auch möglich, einer Variablen, die bislang

eine Zahl abgespeichert hat, eine Zeichenkette zuzuweisen. Python nimmt alle hierfür notwendigen Anpassungen automatisch vor:

```
a = 5
print ("Wert der Variablen a: ",a)
a = "Hallo"
print ("Wert der Variablen a: ",a)
```

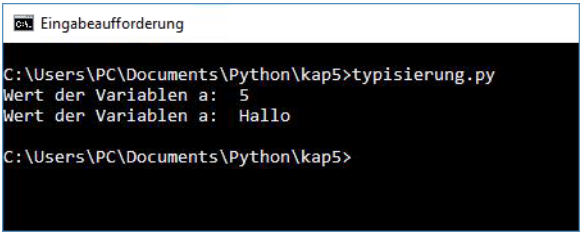

Abbildung 21: Dieselbe Variable kann Zahlen und Zeichenketten aufnehmen

5.5 Datentypen sind auch in Python von Bedeutung

Die dynamische Typisierung erleichtert das Programmieren, da es hierbei in der Regel nicht notwendig ist, sich um die Datentypen zu kümmern. Allerdings gibt es auch einige Ausnahmen, bei denen es dennoch notwendig ist, den Typ der Variablen im Auge zu behalten. Ein Beispiel hierfür ist das erste Programm aus Kapitel 5.3. Dieses forderte den Anwender dazu auf, eine Zahl einzugeben. Danach verdoppelte es den Wert. Allerdings trat hierbei zunächst nicht das gewünschte Ergebnis auf. Da das Programm die Eingabe als Zeichenkette abspeicherte, verdoppelte es die einzelnen Ziffern der Zahl, nicht jedoch ihren Wert.

In diesem Beispiel wurde das Problem durch die Verwendung der `eval`-Funktion gelöst. Es ist jedoch besser, die Umwandlung des Datentyps direkt vorzunehmen. Dazu ist es notwendig, den

gewünschten Datentyp voranzustellen und anschließend die Variable innerhalb einer Klammer hinzuzufügen. Die folgende Tabelle zeigt häufig vorkommende Datentypen:

Datentyp	Variable	Bedeutung
int	integer	ganze Zahlen
float	float	Fließkommazahlen
str	string	Zeichenketten (einzelne Buchstaben, Wörter, Sätze oder Texte)
bool	boolean	Boolesche Variablen/Wahrheitswerte

Wenn man nun die Zeichenkette aus dem Programm aus Kapitel 5.3 ohne die `eval`-Funktion in eine Zahl umwandeln will, ist folgendes Programm notwendig:

```
inhalt = input("Geben Sie bitte eine Zahl ein: ")
inhalt = float(inhalt)*2
print ("Doppelter Wert: ",inhalt)
```

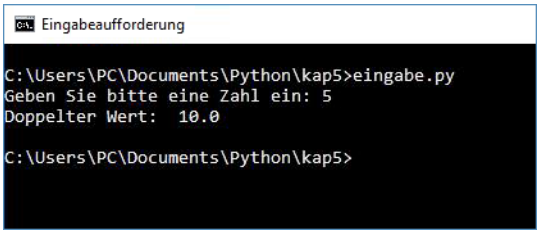

Abbildung 22: Die direkte Umwandlung des Datentyps

Wenn man davon ausgeht, dass der Anwender lediglich ganze Zahlen eingibt, wäre es auch möglich, anstatt des Datentyps `float` den Datentyp `int` zu wählen.

Allerdings ergibt sich bei der direkten Umwandlung des Datentyps ebenfalls ein Problem. Während sich der `eval`-Befehl für die Verarbeitung unterschiedlicher Eingaben eignet, ist es bei der direkten Umwandlung notwendig, den Datentyp bereits von vornherein genau vorzugeben. Wenn der Anwender einen Wert eingibt, der einem anderen Datentyp entspricht, führt das zu einem Fehler. Das hat einen Abbruch des Programms zur Folge.

Obwohl der `eval`-Befehl eigentlich sehr praktisch wäre, wollen wir ihn aufgrund der Sicherheitsprobleme dennoch nicht weiter verwenden. Stattdessen wandeln wir alle Werte direkt in einen anderen Datentyp um, falls dies erforderlich ist. Dabei gehen wir vorerst davon aus, dass der Anwender stets den richtigen Datentyp eingibt. In Kapitel 12 werden Sie dann noch lernen, wie man mit falschen Eingaben richtig umgeht. Dieses Kapitel stellt die Fehlerbehandlung vor, die einen Abbruch des Programms verhindert, selbst wenn der Anwender einen falschen Datentyp eingibt.

Darüber hinaus gibt es verschiedene Funktionen, die einen bestimmten Datentyp voraussetzen. Ein Beispiel hierfür ist die `upper`-Methode. Diese dient dazu, die Kleinbuchstaben in einem Text in Großbuchstaben umzuwandeln:

```
text = "hallo"
print ("Wert der Variablen text vor der Veränderung: ",text)
text = text.upper()
print ("Wert der Variablen text nach der Veränderung: ",text)
```

Diese Methode lässt sich selbstverständlich nur auf Variablen des Typs `str` anwenden, die Text enthalten. Um das Verhalten des Programms auszuprobieren, ist es allerdings sinnvoll, anstatt eines Wortes eine Zahl einzugeben – einmal in Anführungszeichen und einmal ohne.

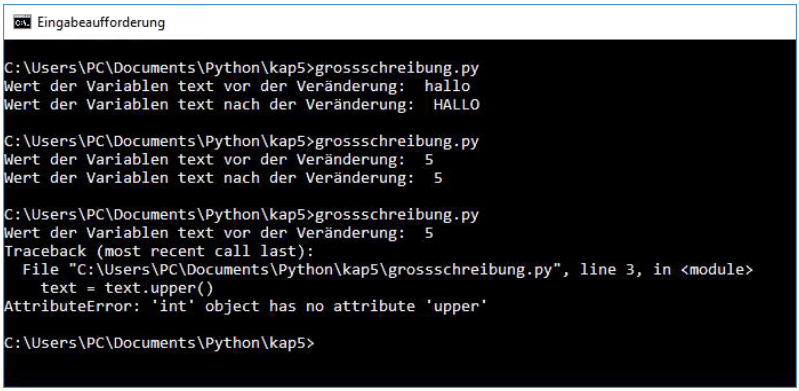

Abbildung 23: Der upper-Befehl mit verschiedenen Variablentypen

Bei der Verwendung eines Wortes erledigt der Befehl seine Aufgabe wie geplant. Wenn man eine Zahl in Anführungszeichen eingibt, wird diese ebenfalls als `str`-Variable gespeichert. Das hat zur Folge, dass sich der `upper`-Befehl ausführen lässt.

Da es zu einer Ziffer jedoch keinen zugehörigen Großbuchstaben gibt, findet keine Veränderung statt. Bei der letzten Ausführung wurde die Zahl ohne Anführungszeichen eingegeben, sodass sie als `int`-Variable abgespeichert wurde. Das führte jedoch zu einer Fehlermeldung, da der `upper`-Befehl nur für `str`-Variablen zulässig ist.

Diese Beispiele zeigen, dass bei vielen Befehlen der Variablentyp eine wichtige Rolle spielt. Aus diesem Grund ist es trotz der dynamischen Typisierung sinnvoll, sich beim Programmieren stets vor Augen zu halten, um welche Datentypen es sich bei den verwendeten Variablen handelt.

Wenn Sie herausfinden wollen, welchen Datentyp eine Variablen besitzt, können Sie den Befehl `type()` verwenden. Dieser gibt den entsprechenden Datentyp zurück. Das zeigt folgendes Programm:

```
a = 3
print ("Datentyp Variable a: ",type(a))
b = 3.563467
print ("Datentyp Variable b: ",type(b))
c = True
print ("Datentyp Variable c: ",type(c))
d = "Hallo"
print ("Datentyp Variable d: ",type(d))
e = (2, 4)
print ("Datentyp Variable e: ",type(e))
```

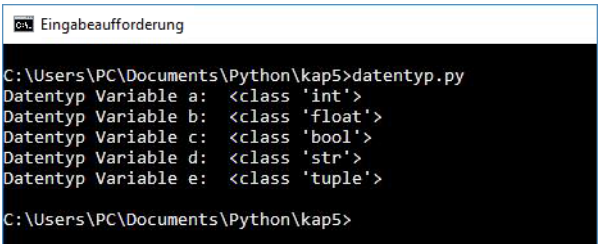

Abbildung 24: Die Ausgabe der verschiedenen Datentypen

5.6 Übungsaufgabe: mit Variablen arbeiten

Aufgabe 1

Schreiben Sie ein Programm, das den Anwender dazu auffordert, zwei Zahlen einzugeben. Speichern Sie diese in zwei unterschiedlichen Variablen. Das Programm soll danach die beiden Werte addieren und ausgeben.

Aufgabe 2

Schreiben Sie ein Programm, das den Anwender dazu auffordert, einen beliebigen Inhalt einzugeben. Um die Eingabe unterschiedlicher Datentypen zu ermöglichen, soll an dieser Stelle nochmals der `eval`-Befehl verwendet werden – obwohl dies normalerweise nicht zu empfehlen ist. Speichern Sie den Wert in einer Variablen und geben Sie anschließend ihren Datentyp aus.

Lösung Aufgabe 1

```
x = float(input("Geben Sie die erste Zahl ein: "))
y = float(input("Geben Sie die zweite Zahl ein: "))
print ("Summe: ", x+y)
```

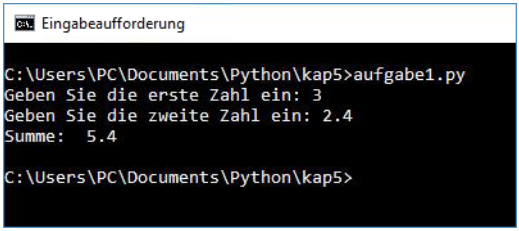

Abbildung 25: Die Ausführung des Programms zu Aufgabe 1

Lösung Aufgabe 2

```
inhalt = eval(input("Geben Sie einen Wert ein: "))
print ("Datentyp: ", type(inhalt))
```

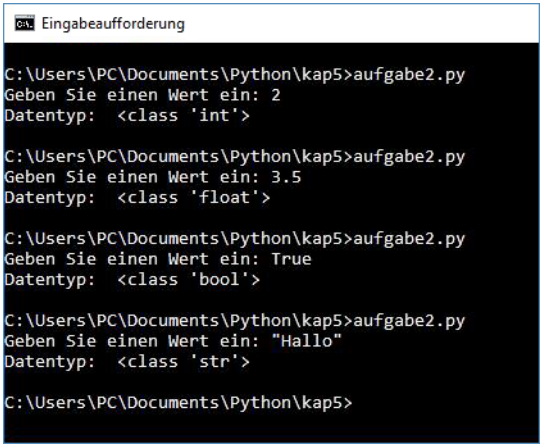

Abbildung 26: Die Ausführung des Programms mit verschiedenen Datentypen

6. Datenstrukturen in Python

In den Übungsaufgaben zu Kapitel 4 wurde das Beispiel eines Baumarkts angesprochen. Wenn man sich überlegt, welche Aufgaben das Verwaltungsprogramm erfüllen soll, dann kommt man sicherlich schnell zu dem Schluss, dass dieses die Eigenschaften eines bestimmten Artikels erfassen muss.

Wenn man beispielsweise eine Bohrmaschine in das Sortiment aufnehmen will, ist es sinnvoll, die Artikelnummer, die Leistungsaufnahme, die Marke und den Preis festzuhalten. Sicherlich gibt es noch viele weitere Details, die zu berücksichtigen sind, wenn man ein derartiges Projekt in die Praxis umsetzt. Der Übersichtlichkeit halber soll sich dieses Beispiel jedoch auf diese vier Werte beschränken.

Selbstverständlich ist es möglich, für die Erfassung der vier Werte jeweils eine eigene Variable zu verwenden. Man könnte beispielsweise folgende Werte in das Programm eingeben:

```
artikelnummer = 10010
leistung = 850
marke = "Bosch"
preis = 59.99
```

Für ein einfaches Programm, das nur die Werte eines einzelnen Produkts aufnimmt, wäre dies sicherlich eine praktikable Vorgehensweise. Allerdings ist davon auszugehen, dass ein Baumarkt nicht nur eine einzige Bohrmaschine anbietet. Wenn mehrere verschiedene Artikel zur Auswahl stehen, wird es bereits deutlich schwieriger, die Werte zu erfassen. Für nur drei Bohrmaschinen wären bereits folgende Eingaben notwendig:

```
artikelnummerBohrmaschine1 = 10010
leistungBohrmaschine1 = 850
markeBohrmaschine1 = "Bosch"
preisBohrmaschine1 = 59.99

artikelnummerBohrmaschine2 = 10014
leistungBohrmaschine2 = 1000
markeBohrmaschine2 = "Makita"
preisBohrmaschine2 = 54.99

artikelnummerBohrmaschine3 = 10003
leistungBohrmaschine3 = 600
markeBohrmaschine3 = "Metabo"
preisBohrmaschine3 = 39.99
```

Wenn man nun davon ausgeht, dass ein durchschnittlicher Baumarkt nicht nur drei, sondern Dutzende Elektrowerkzeuge anbietet, wird schnell deutlich, dass diese Form der Datenerfassung einen riesigen Aufwand mit sich bringt. Es kommt hinzu, dass auch der Zugriff auf die Daten schwierig ist. Jedes Mal, wenn man einen Wert abfragen will, ist es hierfür notwendig, den genauen Variablennamen einzugeben. Das macht eine Automatisierung sehr schwierig.

Wenn man die entsprechenden Werte auf Papier erfasst, verwendet man dafür eine bestimmte Ordnungsstruktur. Beispielsweise ist es möglich, für jeden einzelnen Produkttyp ein eigenes Blatt zu verwenden, um die Übersichtlichkeit zu erhöhen. Darauf befindet sich wiederum eine Liste oder eine Tabelle, in der die Einzelheiten zu den einzelnen Geräten vermerkt sind. Darüber hinaus ist es möglich, übergeordnete Strukturen zu verwenden – beispielsweise indem man Elektrowerkzeuge, Handwerkzeuge, Baustoffe und andere Produkte jeweils in einem eigenen Ordner zusammenfasst.

Python ermöglicht es ebenfalls, Daten zu strukturieren. Das sorgt für eine bessere Übersichtlichkeit und erleichtert den Umgang mit größeren Datensätzen deutlich. Diese Programmiersprache zeichnet sich dadurch aus, dass sie sehr vielfältige Datenstrukturen anbietet, die dem Programmierer viele Freiheiten lassen.