Kapitel 5
Variablen: unverzichtbar für die Programmierung mit Python

Die bisherigen Programme dienten lediglich dazu, einen Text auszugeben oder einfache mathematische Berechnungen durchzuführen. Dafür ist es jedoch eigentlich nicht notwendig, ein eigenes Programm zu verfassen, da es dafür bereits viele weitere Möglichkeiten gibt – beispielsweise Präsentationsprogramme oder Taschenrechner. Ein Computerprogramm dient hingegen dazu, feste Muster für Berechnungen vorzugeben und die Ergebnisse zu speichern und für weitere Prozesse zu verwenden.

Um die Werte zu erfassen, sind Variablen notwendig. Diese dienen als Speicher für ein ganz bestimmtes Ergebnis. Sie sind so wichtig, dass es kaum ein Computerprogramm gibt, das ohne Variablen auskommt.

In Kapitel 3 wurden bei der interaktiven Anwendung des Python-Prompts bereits die ersten Variablen verwendet – ohne jedoch auf deren genaue Funktionsweise einzugehen. Das wird in diesem Kapitel nachgeholt.

5.1 Die Aufgabe von Variablen in einem Computerprogramm

Wenn ein Computerprogramm einen bestimmten Wert speichern soll, dann muss es diesen im Arbeitsspeicher ablegen. Dieser besteht aus Tausenden elektronischen Informationen – den einzelnen Bits. Wenn man ein Computerprogramm schreibt, ist es allerdings nicht notwendig, sich auf dieser Ebene mit der Datenspeicherung zu befassen. Anstatt dessen kommen Variablen zum Einsatz. Diese ermöglichen es, einen Namen für die entsprechende Information vorzugeben und daraufhin zu jedem beliebigen Zeitpunkt auf sie zuzugreifen – ganz einfach unter Nennung des selbst

vorgegebenen Namens. Um die Funktionsweise der Variablen zu verstehen, ist es jedoch sinnvoll, sich mit der Speicherorganisation des Computers zu befassen.

Man kann sich den Arbeitsspeicher eines Computers wie einen großen Setzkasten vorstellen. Dieser verfügt über viele kleine Fächer, die unterschiedliche Gegenstände aufnehmen können – im übertragenen Sinn die Werte, die das Programm abspeichern soll.

Wenn man zu einem späteren Zeitpunkt einen bestimmten Gegenstand benötigt, ist es notwendig, zu wissen, in welchem Fach er sich befindet. Bei einem kleinen Setzkasten erinnert sich der Anwender in der Regel, wo er das entsprechende Objekt abgelegt hat. Bei einem sehr großen Kasten kann es dabei jedoch bereits zu Verwechslungen kommen. Daher ist es sinnvoll, die einzelnen Fächer zu beschriften. Dadurch ist es möglich, anhand der Bezeichnung den benötigten Gegenstand schnell zu finden. Eine entsprechende Organisationsstruktur ist auch bei der Verwendung von Variablen sehr wichtig – insbesondere wenn man berücksichtigt, dass ein gewöhnlicher Arbeitsspeicher Platz für viele Tausend Variablen bietet. Daher ist es auch hier notwendig, einen festen Namen zu vergeben. Das Programm hinterlegt bei der Ausführung automatisch den zugehörigen Speicherort, sodass die benötigten Informationen unmittelbar zur Verfügung stehen.

Der Variablenname dient dazu, einen bestimmten Speicherort eindeutig zu identifizieren. Hierfür ist es erforderlich, dass er nicht doppelt verwendet wird, da das sonst zu Verwechslungen führen könnte. Sollte im weiteren Verlauf eine Bezeichnung zum Einsatz kommen, die bereits früher verwendet wurde, kann das einen Fehlern erzeugen.

Die Objekte in einem Setzkasten können eine unterschiedliche Größe haben. Daher gibt es spezielle Ausführungen, bei denen sich die seitlichen Abgrenzungen verschieben lassen, um den verfügbaren Raum anzupassen. Ähnliches gilt für die Variablen in einem Computerprogramm. Diese können eine kleine Zahl oder ein komplettes Buch aufnehmen. Der dafür benötigte

Speicherplatz kann sehr unterschiedlich sein. In vielen Programmiersprachen ist es daher notwendig, die Variablentypen bereits zu Beginn fest vorzugeben. So reserviert das Programm einen passenden Speicherplatz. Python erledigt diese Aufgabe jedoch je nach Bedarf während der Ausführung. Daher können die Variablen hier unterschiedliche Größen annehmen und sind nicht an spezielle Typen gebunden.

5.2 Variablen in Python verwenden

Bereits in Kapitel 3 wurden bei der interaktiven Nutzung von Python die ersten Variablen verwendet. Dazu wurde der Variablenname gefolgt von einem Gleichheitszeichen und von einer Zahl eingegeben: `a = 5`. Auch wenn man ein Programm in eine eigene Datei schreibt, verläuft die Verwendung der Variablen nach dem gleichen Muster.

Zunächst ist es notwendig, einen Namen für die Variable festzulegen. Dieser kann beliebige Buchstaben, Ziffern und den Unterstrich enthalten. Dabei ist es wichtig, zu beachten, dass am Anfang des Variablennamens keine Ziffer stehen darf – lediglich Buchstaben oder Unterstriche. Das bedeutet, dass Ausdrücke wie `variable_1` oder `_1_variable` erlaubt sind, `1_variable` hingegen nicht.

Darüber hinaus ist es wichtig, die Groß und Kleinschreibung zu beachten. Python wertet die Ausdrücke `Variable`, `variable` und `VARIABLE` jeweils als unterschiedliche Bezeichnungen und legt daher drei verschiedene Variablen dafür fest. Bei neueren Python Versionen (ab Version 3) werden Unicode-Zeichen unterstützt. Das bedeutet, dass die Variablennamen auch Umlaute, griechische oder russische Buchstaben sowie viele weitere Zeichen enthalten können. Das ist jedoch sehr unüblich und sollte vermieden werden.

Nach dem selbst gewählten Variablennamen folgt das Gleichheitszeichen. Dieses wird in diesem Fall auch als Zuweisungsoperator bezeichnet. Es dient dazu, der Variablen einen bestimmten Wert zuzuweisen.

Danach folgt der Wert, den die Variable annehmen soll. Dabei kann es sich um ganze Zahlen, um Kommazahlen, um Buchstaben oder um ganze Wörter und Sätze handeln. Auch Wahrheitswerte (`True` und `False`) sind möglich. Dabei muss der Programmierer den Datentyp nicht vorgeben. Python erkennt anhand der Zuweisung automatisch, um welche Art von Information es sich dabei handelt und reserviert einen passenden Speicherplatz. Bei der Verwendung von Buchstaben, Wörtern oder anderen Zeichenketten ist es wichtig, die Werte in Anführungszeichen zu setzen. Geschieht das nicht, geht Python davon aus, dass es sich um einen anderen Variablen- oder Funktionsnamen handelt. Wenn dieser jedoch nicht definiert wurde, führt das zu einer Fehlermeldung.

Bei der interaktiven Ausführung von Python war es lediglich notwendig, den Variablennamen einzutippen und die Eingabetaste zu betätigen, um den Wert abzufragen. Wenn man das Programm in einer eigenen Datei verfasst, ist dies jedoch nicht ausreichend. In diesem Fall muss der Variablennamen in einem print-Befehl stehen – allerdings ohne Anführungszeichen:

```
print (a)
```

Ein einfaches Programm, das eine Variable bestimmt, ihr einen Wert zuweist und diesen anschließend ausgibt, könnte folgendermaßen aussehen:

```
a = 5
print (a)
```

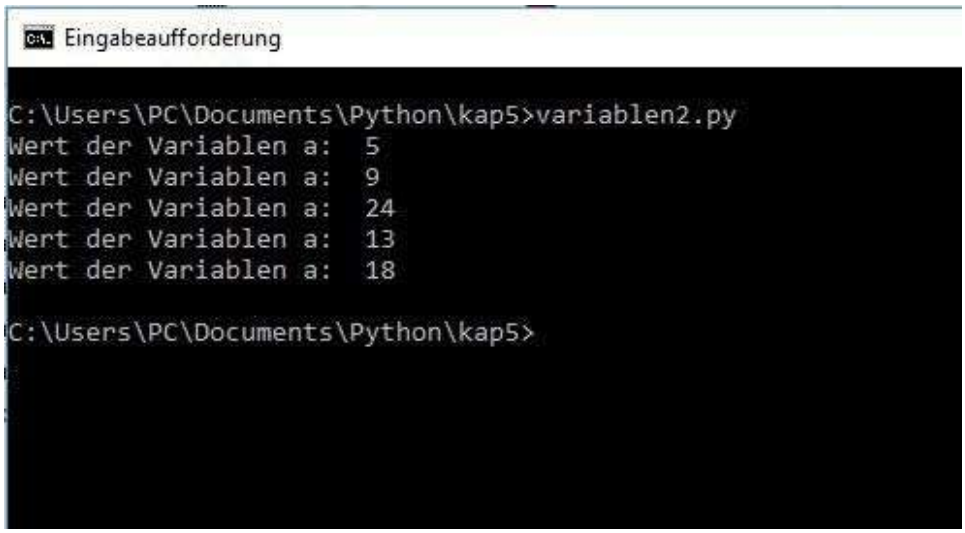
Dabei ist es auch möglich, bei der Ausgabe einen Text mit einer Variablen zu verbinden. Dafür ist es lediglich notwendig, die Bestandteile durch ein Komma voneinander zu trennen:

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

Screenshot 14 Die Ausgabe des Werts der Variablen

Im weiteren Verlauf kann die Variable beliebige weitere Werte annehmen. Dabei kann man einen einfachen Wert zuweisen, eine Rechenaufgabe ausführen oder den Wert einer anderen Variablen verwenden:

```
a = 5
print ("Wert der Variablen a: ",a)
a = 9
print ("Wert der Variablen a: ",a)
a = 3 * 8
print ("Wert der Variablen a: ",a)
b = 13
a = b
print ("Wert der Variablen a: ",a)
a = a + 5
print ("Wert der Variablen a: ",a)
```



Screenshot 15 Die Variable a nimmt verschiedene Werte an.

Von besonderer Bedeutung ist dabei die letzte Zuweisung: `a = a + 5`. Wenn man diesen Ausdruck unter mathematischen Gesichtspunkten betrachtet, ist er eindeutig falsch, da der Wert der Variablen `a` nicht gleich ihrem Wert plus fünf sein kann. Daran wird deutlich, dass es sich hierbei nicht um das Gleichheitszeichen im mathematischen Sinn handelt. Es dient hier vielmehr als ein Befehl, der der Variablen, die links vom Gleichheitszeichen steht, den Wert, der auf dessen rechter Seite steht, zuweist. Dieser kann auch auf den bisherigen Wert der Variablen Bezug nehmen. Der Ausdruck `a = a + 5` bedeutet daher, dass der Variablen `a` ihr bisheriger Wert plus fünf zugewiesen werden soll.

5.3 Den Wert einer Variablen durch eine Eingabe des Nutzers festlegen

Einer der großen Vorteile eines Computerprogramms besteht darin, dass es mit dem Nutzer interagieren und die Berechnungen an dessen Anforderungen anpassen kann. Bislang haben die Programme jedoch nur Werte berechnet, die bereits beim Erstellen des Programms vorgegeben wurde. Eine Interaktion mit dem Anwender fand nicht statt.

Der nächste Schritt besteht daher darin, den Nutzer in das Programm einzubeziehen. Um zu erläutern, wie das funktioniert, soll ein Programm erstellt werden, das den Anwender dazu auffordert, eine Zahl einzugeben. Danach berechnet es den doppelten Wert dieser Zahl und gibt diese auf dem Bildschirm aus.

Dazu ist es zunächst notwendig, nach bekanntem Muster einen kurzen Text mit der Eingabeaufforderung zu schreiben. Für die Eingabe der Zahl ist folgende Zeile notwendig:

```
inhalt = input()
```

Diese erzeugt zunächst die Variable inhalt. Ihr wird als Wert die Funktion `input()` zugewiesen. Diese gibt die Eingabe des Anwenders zurück. Danach ist es notwendig, den Wert zu verdoppeln und ihn anschließend auszugeben:

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

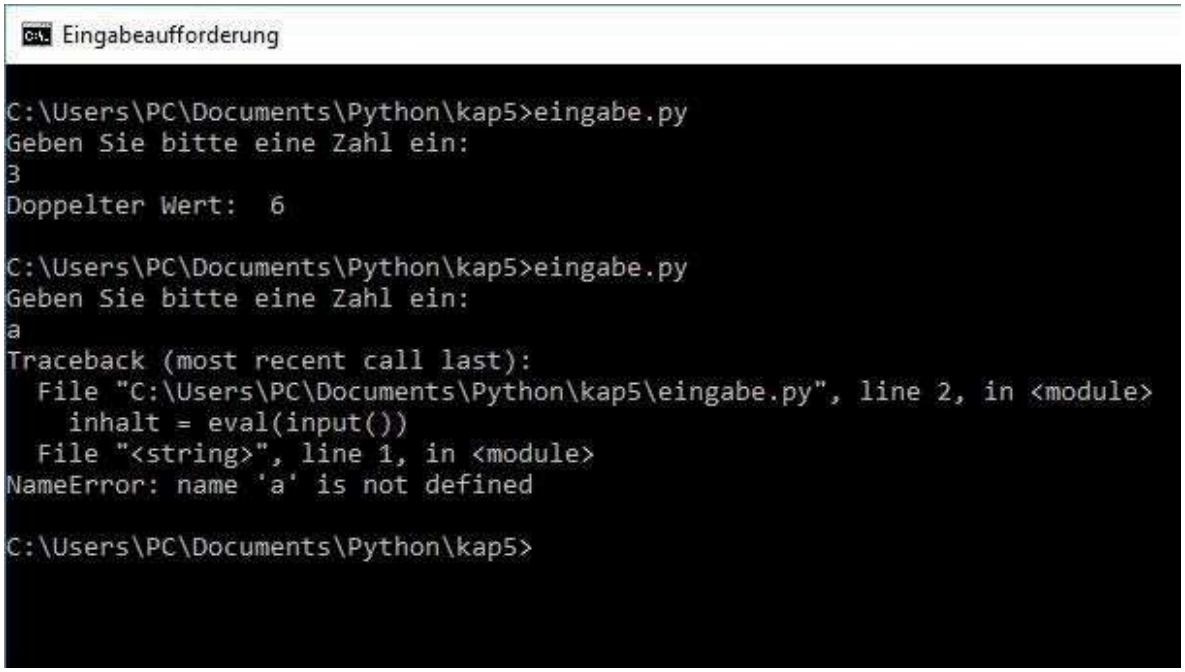
Screenshot 16 Die Ausgabe des Programms

Wenn man sich die Ausgabe genau betrachtet, fällt auf, dass dieses keinen Sinn ergibt. Der doppelte Wert von 3 wäre 6 und nicht 33. Dieses seltsame Ergebnis liegt daran, dass der Input-Befehl die Werte stets als Zeichen zurückgibt und nicht als Zahl. Die Verdopplung eines Zeichens führt dazu, dass dieses lediglich zwei Mal hintereinander ausgegeben wird. Die Verdopplung des Zeichens a wäre demnach aa. Analog dazu ist 33 die Verdopplung des Zeichens 3.

Um das gewünschte Ergebnis zu erzielen, ist der Befehl `eval()` notwendig. Dieser speichert das Ergebnis unter dem passenden Datentyp ab. Der input-Befehl steht dabei innerhalb der Klammer. Demnach muss das Programm wie folgt abgeändert werden:

```
print ("Geben Sie bitte eine Zahl ein: ")
inhalt = eval(input())
inhalt = inhalt*2
```

```
print ("Doppelter Wert: ",inhalt)
```



Screenshot 17 Die richtige Verdopplung und die Fehlermeldung bei Eingabe eines Buchstabens

Anstelle des `eval`-Befehls wäre es auch möglich, die Eingabe auf direktem Wege in einen anderen Datentyp zu überführen. Wie das funktioniert, wird in Kapitel 5.5 erklärt. Dabei wäre es jedoch notwendig, den Datentyp bereits von Beginn an genau vorzugeben. Wenn der Anwender einen Wert eingibt, der einen anderen Typ erfordert, führt das zu einem Fehler. Der `eval`-Befehl eignet sich hingegen für viele verschiedene Alternativen und erkennt selbstständig, wie er die Eingabe umwandeln muss. Lediglich bei Buchstaben, Wörtern und anderen Zeichenketten ist Vorsicht geboten. Damit diese richtig abgespeichert werden, ist es notwendig, dass der Anwender sie in Anführungszeichen setzt. Ohne diese kommt es zu einer Fehlermeldung.

Manche Anwender haben sich vielleicht gefragt, weshalb nach dem `input`-Befehl eine leere Klammer notwendig ist. Das liegt daran, dass es sich hierbei um eine Funktion handelt, der ein Wert übergeben werden kann. Dabei handelt es sich um die Eingabeaufforderung für den Nutzer, die bislang in einem eigenen `print`-Befehl steht. Anstatt dessen ist es möglich, den Text

einfach in die Klammer einzufügen. Das vereinfacht nicht nur den Programmcode, darüber hinaus entsteht auf diese Weise eine ansprechendere Darstellung, bei der die Eingabe in der gleichen Zeile wie die Aufforderung erfolgt:

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

Screenshot 18 Das Programm mit der Eingabeaufforderung im input-Befehl

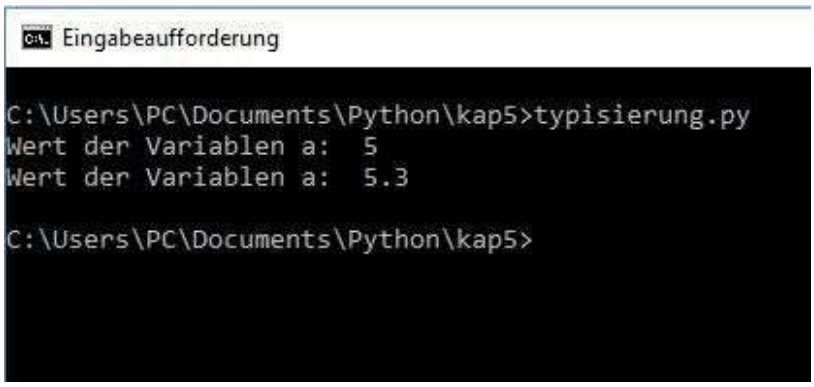
5.4 Dynamische Typisierung: viele Freiheiten bei der Nutzung von Variablen

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 Variable 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. Bei Python stellt dies jedoch kein Problem dar:

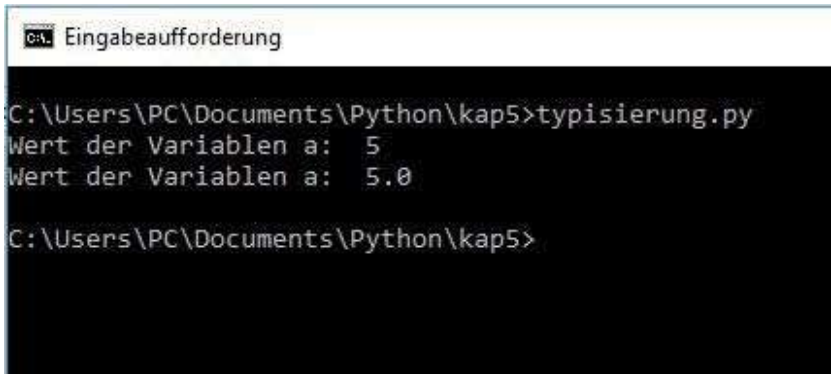
```
a = 5
print ("Wert der Variablen a: ",a)
a = 5.3
print ("Wert der Variablen a: ",a)
```



Screenshot 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)
```

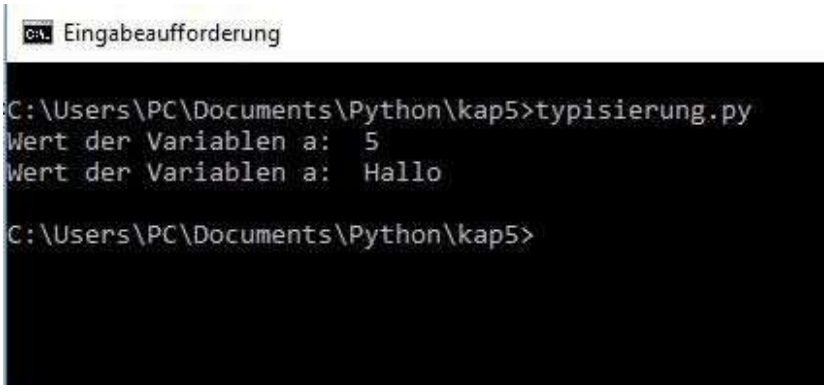


Screenshot 20 Die Ausgabe als Ganzzahl und als Fließkommazahl

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)
```



Screenshot 21 Die gleiche 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 Nutzer 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 auch möglich, 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:

```
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, wäre folgendes Programm notwendig:

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

Screenshot 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.

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 vom Typ `str` anwenden, die Text enthalten. Um das Verhalten des Programms auszuprobieren, ist es allerdings sinnvoll, anstatt eines Worts eine Zahl einzugeben – einmal in Anführungszeichen und einmal ohne.

Screenshot 23 Der upper-Befehl mit verschiedenen Variablentypen

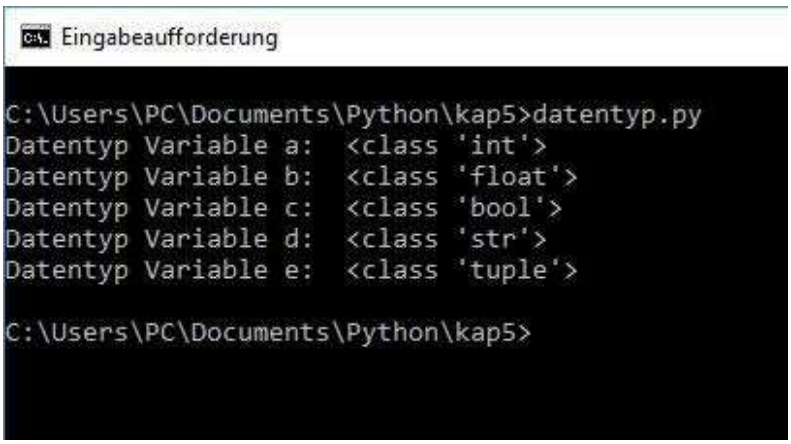
Bei der Verwendung eines Worts erledigt der Befehl seine Aufgabe wie geplant. Wenn man eine Zahl in Anführungszeichen eingibt, wird diese

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 welchen Datentyp es sich bei den verwendeten Variablen handelt.

Wenn man herausfinden will, welchen Datentyp eine Variablen beinhaltet, kann man 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))
```



Screenshot 24 Die Ausgabe der verschiedenen Datentypen

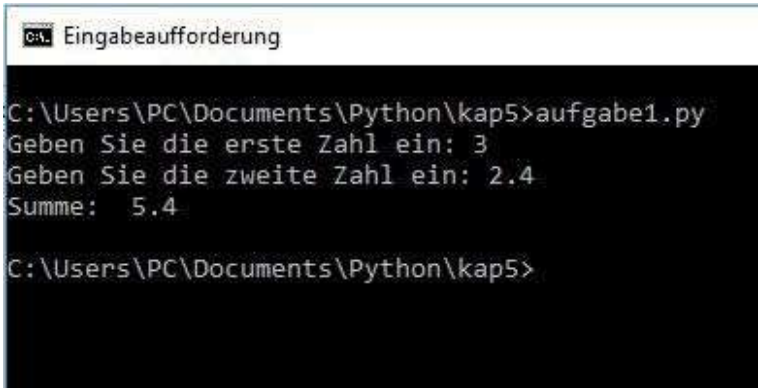
5.6 Übungsaufgabe: Mit Variablen arbeiten

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.
2. Schreiben Sie ein Programm, das den Anwender dazu auffordert, einen beliebigen Inhalt einzugeben. Speichern Sie den Wert in einer Variablen und geben Sie anschließend ihren Datentyp aus.

Lösungen:

1.

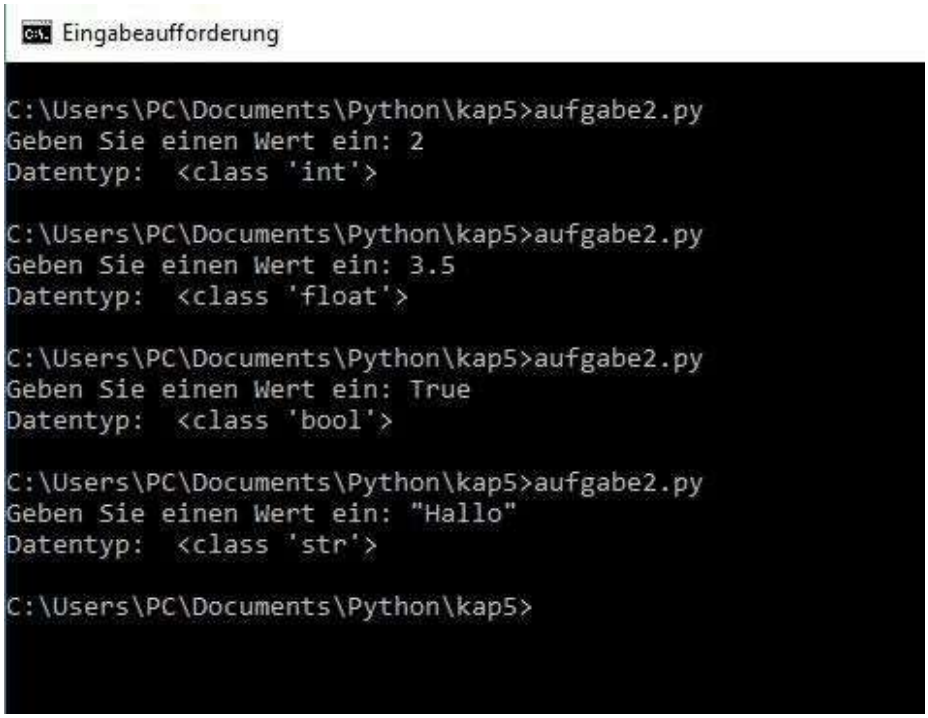
```
x = eval(input("Geben Sie die erste Zahl ein: "))
y = eval(input("Geben Sie die zweite Zahl ein: "))
print ("Summe: ", x+y)
```



Screenshot 25 Die Ausführung des Programms zu Aufgabe 1

2.

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



Screenshot 26 Die Ausführung des Programms mit verschiedenen Datentypen