Kapitel 7
Entscheidungen im Programm treffen

In den vorhergehenden Kapiteln wurden einfache Ausgabebefehle, Variablen und Datenstrukturen behandelt. Diese ermöglichen es, verschiedene Werte zu berechnen und auszugeben. Für derartige Aufgaben wäre es jedoch eigentlich nicht notwendig, ein eigenes Computerprogramm zu erstellen. Mit einem gängigen Tabellenkalkulationsprogramm wäre diese Aufgabe deutlich einfacher zu erledigen.

Eines der wesentlichen Merkmale eines Computerprogramms besteht darin, dass es auf bestimmte Ereignisse oder Eingabewerte reagieren kann. Das bedeutet, dass der Ablauf des Programms nicht immer gleich ist, sondern von den aktuellen Werten des Programms abhängt.

Um den Ablauf des Programms zu bestimmen, kommen `if`-Abfragen zum Einsatz. Diese beinhalten eine bestimmte Bedingung. Der Programmteil, der zur `if`-Abfrage gehört, wird nur dann ausgeführt, wenn die Bedingung erfüllt ist. Trifft das nicht zu, wird er übersprungen. Auf diese Weise lässt sich der Ablauf des Programms an bestimmte Werte koppeln. Die `if`-Abfrage stellt ein sehr wichtiges Instrument beim Programmieren dar und kommt in fast allen Programmen vor.

7.1 Der Schlüsselbegriff if

Beim Aufstellen einer Bedingung in der gewöhnlichen Sprache kommt normalerweise das Begriffspaar “wenn-dann” zum Einsatz. Ein Beispiel hierfür wäre: “Wenn der Schüler die Aufgabe richtig gelöst hat, dann erhält er eine gute Note.” Die Programmiersprache Python orientiert sich ebenfalls

an dieser natürlichen Satzstruktur. An erster Stelle steht stets der Schlüsselbegriff `if` – also die englische Übersetzung für “wenn”. Danach ist es notwendig, die Bedingung anzugeben. Anstelle des Begriffs “dann” folgt hingegen ein Doppelpunkt.

Die Befehle, die zu erledigen sind, stehen nach dem Doppelpunkt. Es ist möglich, einen Befehl direkt hinter den Doppelpunkt zu schreiben. Das ist jedoch nicht üblich und auch nicht zu empfehlen. Der Grund dafür liegt darin, dass oftmals nicht nur ein einzelner Befehl ausgeführt werden muss, sondern mehrere. Wenn der erste Befehl jedoch direkt hinter der Bedingung steht, ist es nicht möglich, weitere Befehle hinzuzufügen. Daher ist es sinnvoll, ihn stets in eine neue Zeile zu schreiben.

Wenn beim Eintreffen einer Bedingung mehrere Befehle ausgeführt werden sollen, dann ist es notwendig, den Anfangs- und den Endpunkt zu markieren. Nur so erkennt das Programm, welche Befehle mit dieser Bedingung verknüpft sind und welche nicht. Die meisten Programmiersprachen verwenden zu diesem Zweck eine geschweifte Klammer. Zusätzlich rücken die Programmierer den zusammengehörigen Block ein. Das ist für die Funktionalität des Programms bei der Mehrheit der Programmiersprachen zwar nicht notwendig, doch führt diese Vorgehensweise zu einem deutlich übersichtlicheren Code. Wie bereits in der Einleitung erwähnt, bestand ein Grundgedanke bei der Entwicklung von Python darin, möglichst auf Klammern zu verzichten. Daher wird hier ein anderes Prinzip verwendet: Hier markieren die Einrückungen, die in den meisten anderen Programmiersprachen keinen Einfluss auf die Funktion des Programms haben, den Anfang und das Ende des entsprechenden Blocks. Alle Befehle, die zur `if`-Abfrage gehören, müssen daher eingerückt werden. Wenn danach Befehle folgen, die unabhängig von der Bedingung ausgeführt werden sollen, müssen diese ohne Einrückung hinzugefügt werden. Daraus ergibt sich für die `if`-Abfrage folgende Struktur:

```
if Bedingung:
  Befehl 1
```

```
  Befehl 2
  .
  .
  .
  Befehl n
Befehle, die unabhängig von der Bedingung sind
```

Um eine Einrückung zu erzeugen, kommen verschiedene Möglichkeiten infrage. Sehr beliebt ist es, die Tabulatortaste zu verwenden. Diese erzeugt einen Freiraum, der stets eine identische Größe aufweist. Alternativ dazu ist es auch erlaubt, Leerzeichen zu verwenden. In der Regel kommen zu diesem Zweck vier Leerzeichen zum Einsatz. Dieser Wert ist jedoch nicht zwingend vorgegeben. Bei stark verzweigten Programmen mit vielen Einrückungen ist es beispielsweise sinnvoll, nur zwei Leerzeichen zu verwenden, damit der Code nicht zu stark auf die rechte Seite wandert. Manche Programmierer verwenden auch acht Leerzeichen, um die Einrückungen besonders deutlich zu machen.

Welche dieser Möglichkeiten zum Einsatz kommt, hängt von den Vorlieben des Programmierers ab. Es ist jedoch wichtig, sich zu Beginn für eine von ihnen zu entscheiden und diese dann konsequent beizubehalten. Wenn man die Form der Einrückungen mischt, kann das zu Fehlern führen.

7.2 Vergleiche: wichtig für das Aufstellen der Bedingung

Die grundlegende Struktur der `if`-Abfrage ist nun bekannt, doch ist es bislang noch nicht möglich, ein Programm mit diesem Element zu verfassen. Der Grund dafür liegt darin, dass noch nicht behandelt wurde, wie die Bedingung bei der `if`-Abfrage aufgestellt wird.

Die einfachste Möglichkeit besteht sicherlich darin, einen Wert auf Gleichheit zu einem anderen Wert zu überprüfen. Wenn man einen bestimmten Teil des Programms nur dann ausführen will, wenn der Wert der Variablen `a` der Zahl 5 entspricht, dann wäre hierfür folgende Bedingung notwendig:

```
a == 5
```

Auffällig ist, dass hierbei ein doppeltes Gleichheitszeichen zum Einsatz kommt. Das liegt daran, dass dem einfachen Gleichheitszeichen in Python bereits eine andere Funktion zukommt: als Zuweisungsoperator. Wenn man nur ein Gleichheitszeichen verwendet, dann würde das bedeuten, dass man der Variablen `a` den Wert 5 zuweist und nicht überprüft, ob er diesem Wert entspricht.

Mit diesen Informationen ist es bereits möglich, ein erstes Programm mit einer `if`-Abfrage zu gestalten:

```
a = eval(input("Geben Sie bitte die Zahl 5 ein: "))
if a == 5:
    print("Die Eingabe ist richtig.")
```

Screenshot 36 Die Ausführung mit einer richtigen und mit einer falschen Eingabe

Dieses Programm fordert den Anwender dazu auf, die Zahl 5 einzugeben und speichert den Wert in der Variablen `a` ab. Daraufhin überprüft es den Wert der Variablen und gibt eine Erfolgsmeldung aus, wenn der Nutzer den richtigen Wert eingegeben hat. Bei einem anderen Wert führt das Programm keine Aktion durch.

Die Überprüfung der Gleichheit funktioniert nicht nur bei Zahlen, sondern auch bei Zeichenketten. Es wäre genauso gut möglich, den Anwender dazu

aufzufordern, ein Wort einzugeben und dieses dann auf die Gleichheit zu überprüfen:

```
a = input('Geben Sie bitte das Wort "Hallo" ein: ')
if a == "Hallo":
   print("Die Eingabe ist richtig.")
```

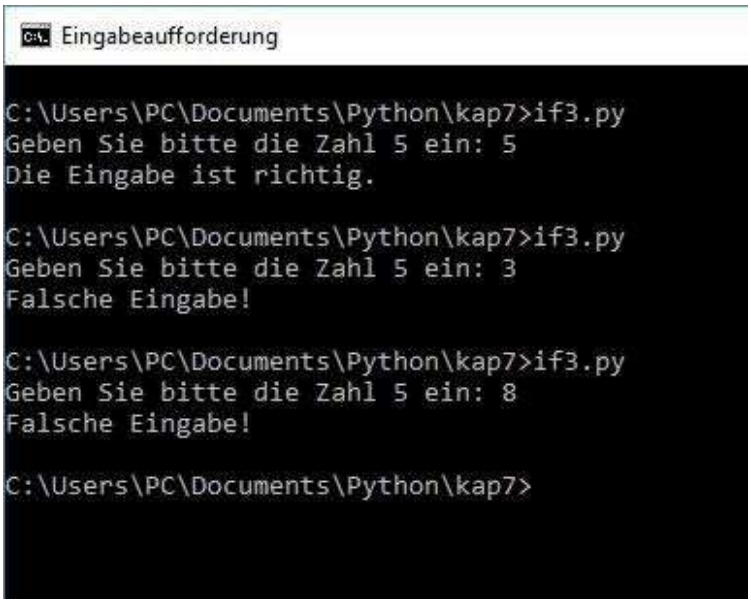
Anmerkung: In diesem Programm wird der `eval`-Befehl weggelassen, da davon auszugehen ist, dass der Anwender eine Zeichenkette eingibt. Mit dem `eval`-Befehl müsste er diese in Anführungszeichen setzen.

Häufig ist es nicht nur notwendig, zu überprüfen, ob eine Zahl gleich wie ein bestimmter Wert ist, sondern ob sie größer oder kleiner als dieser ist. Hierfür kommen folgende Vergleichsoperatoren zum Einsatz:

```
> ist größer
< ist kleiner
>= ist größer oder gleich
<= ist kleiner oder gleich
```

Mit diesen Operatoren ist es möglich, auch eine passende Ausgabe anzugeben, wenn der Anwender die Zahl nicht richtig eingibt. Hierfür sind zwei weitere `if`-Abfragen notwendig – eine die überprüft, ob der Wert größer als 5 ist und die andere, die ausgeführt wird, wenn der Wert kleiner als 5 ist:

```
a = eval(input("Geben Sie bitte die Zahl 5 ein: "))
if a == 5:
   print("Die Eingabe ist richtig.")
if a > 5:
   print("Falsche Eingabe!")
if a < 5:
   print("Falsche Eingabe!")
```



Screenshot 37 Jetzt erhält der Anwender auch bei einem falschen Wert eine Rückmeldung.

Drei verschiedene `if`-Abfragen in das Programm zu integrieren, ist relativ aufwendig. Es geht auch einfacher. Es gibt einen Vergleichsoperator, der die Werte auf Ungleichheit überprüft: `!=`. Mit diesem lassen sich die letzten beiden `if`-Abfragen zu einem Befehl zusammenfassen:

```
a = eval(input("Geben Sie bitte die Zahl 5 ein: "))
if a == 5:
   print("Die Eingabe ist richtig.")
if a != 5:
   print("Falsche Eingabe!")
```

Das Verhalten dieses Programms ist vollkommen identisch zum vorhergehenden Beispiel. Allerdings ist es deutlich kürzer.

Darüber hinaus gibt es noch eine weitere Möglichkeit, um eine Bedingung zu formulieren: mit booleschen Variablen. Diese geben wie bereits zuvor erläutert Wahrheitswerte wieder und können die Werte `True` und `False` annehmen. Sie können einfach in die Bedingung geschrieben werden: Wenn der Wert `True` ist, wird der anschließende Block ausgeführt, wenn er False ist, hingegen nicht.

```
a = True
if a:
   print("Der Wert der Variablen ist True.")
```

Darüber hinaus gibt es einen Verneinungsoperator. Dieser entspricht dem Begriff `not`. Die Bedingung ist in diesem Fall dann erfüllt, wenn das entsprechende Ereignis nicht eintritt. Er lässt sich auf boolesche Variablen (`if not a`) und auf Vergleiche (`if not a == 5`) anwenden.

7.3 Die Verknüpfung mehrerer Bedingungen

In vielen Fällen soll ein bestimmter Programmteil nur dann ausgeführt werden, wenn zwei Bedingungen gleichzeitig erfüllt sind. In diesem Fall wäre es selbstverständlich möglich, zwei `if`-Abfragen ineinander zu schachteln. Doch würde das die Struktur des Programms deutlich komplizierter gestalten. Daher ist es sinnvoll, die beiden Bedingungen direkt miteinander zu verknüpfen. Das ist mit dem Schlüsselbegriff and möglich:

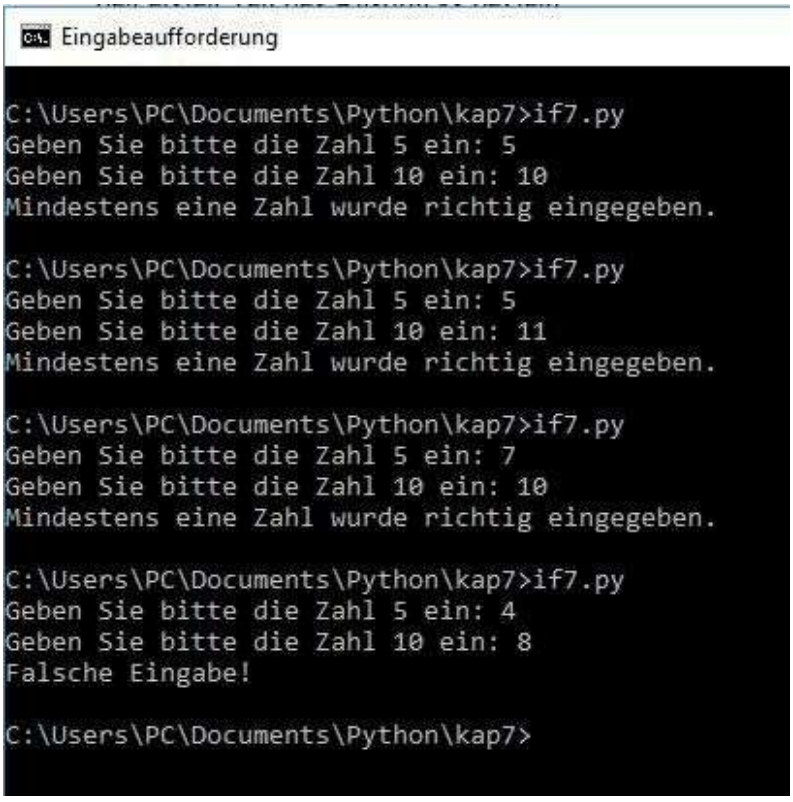
```
a = eval(input("Geben Sie bitte die Zahl 5 ein: "))
b = eval(input("Geben Sie bitte die Zahl 10 ein: "))
if a == 5 and b == 10:
   print("Die Eingabe ist richtig.")
if not (a == 5 and b == 10):
   print("Falsche Eingabe!")
```

Screenshot 38 Die Bedingung ist nur dann erfüllt, wenn beide Zahlen richtig sind.

Um zu überprüfen, ob die Eingabe falsch ist, wird einfach der komplette Ausdruck durch `not` verneint. Dabei ist es wichtig, diesen in eine Klammer zu setzen, da sich dieser Operator sonst nur auf den ersten Teil des Ausdrucks bezieht.

In anderen Fällen ist es ausreichend, wenn eine von mehreren möglichen Bedingungen erfüllt wird. Auch hierfür gibt es einen passenden Operator: `or`. Dieser führt dazu, dass die Bedingung erfüllt ist, wenn eine der beiden Teilbedingungen oder alle beide erfüllt sind:

```
a = eval(input("Geben Sie bitte die Zahl 5 ein: "))
b = eval(input("Geben Sie bitte die Zahl 10 ein: "))
if a == 5 or b == 10:
   print("Mindestens eine Zahl wurde richtig eingegeben.")
if not (a == 5 or b == 10):
   print("Falsche Eingabe!")
```



Screenshot 39 Bei der Verwendung von or ist es ausreichend, dass eine Bedingung erfüllt wird.

Es ist möglich, beliebig viele Teilbedingungen mit den Ausdrücken `and` und `or` zu kombinieren. Auf diese Weise lassen sich sehr komplexe Bedingungen definieren. Dabei ist es wichtig, auf die Klammersetzung zu achten. Wenn man keine Klammern setzt, arbeitet Python den Ausdruck einfach von links nach rechts ab.

7.4 Mit else und elif weitere Alternativen hinzufügen

In den vorhergehenden Abschnitten trat bereits mehrmals das Problem auf, dass das Programm eine bestimmte Aktion durchführen sollte, wenn die Bedingung erfüllt war und eine andere, wenn sie nicht zutraf. Um dieses Verhalten zu erreichen, war es im ersten Versuch notwendig, sich zu überlegen, wie das Gegenteil der entsprechenden Bedingung aussieht. Das ist insbesondere bei komplizierten Bedingungen nicht ganz einfach und bringt außerdem einen erheblichen Aufwand mit sich. Im zweiten Versuch wurde

die Bedingung vollständig mit dem Begriff `not` verneint. Das gestaltete die Aufgabe zwar bereits etwas einfacher, doch brachte auch diese Alternative einigen Aufwand mit sich. Da eine derartige Anforderung jedoch bei unzähligen Programmen auftritt, bietet Python genau wie fast alle anderen Programmiersprachen noch eine weitere Möglichkeit: `else`. Mit diesem einfachen Schlüsselbegriff ist es möglich, einen weiteren Block einzufügen, der nur dann ausgeführt wird, wenn die Bedingung nicht zutrifft. Die Struktur sieht demnach folgendermaßen aus:

```
if Bedingung:
  Befehle, die ausgeführt werden, wenn die Bedingung zutrifft.
else:
  Befehle, die ausgeführt werden, wenn die Bedingung nicht zutrifft.
```

Mit diesem neuen Befehl lässt sich das Programm, das die Richtigkeit einer Eingabe überprüft, noch etwas einfacher gestalten:

```
a = eval(input("Geben Sie bitte die Zahl 5 ein: "))
if a == 5:
   print("Die Eingabe ist richtig.")
else:
   print("Falsche Eingabe!")
```

Häufig tritt auch der Fall auf, dass ein Programm mehr als 2 Optionen anbieten soll – je nachdem, welchen Wert eine Variable hat. Auch hierbei wäre es möglich, das Programm mit einfachen `if`-Abfragen zu gestalten. Doch wäre dies nicht nur sehr aufwendig. Darüber hinaus wäre es notwendig, die Abfragen ineinander zu schachteln. Das führt zu einer sehr komplizierten Struktur und damit zu einer erhöhten Fehleranfälligkeit. Daher bietet Python auch hierfür eine praktische Möglichkeit an: `elif`. Diese erlaubt es, einen weiteren Block hinzuzufügen und diesen an eine weitere Bedingung zu knüpfen. Der Inhalt wird ausgeführt, wenn die vorherigen Bedingungen alle nicht erfüllt sind, wenn die für diesen Block aufgestellte Bedingung hingegen zutrifft. Es ist dabei möglich, beliebig viele `elif`-Blöcke einzufügen. Anschließend kann ein `else`-Block stehen. Dieser wird ausgeführt, wenn keine der vorherigen Bedingungen zutrifft. Dieser Block ist jedoch nicht

zwingend erforderlich. Die Struktur sieht wie folgt aus:

```
if Bedingung 1:
  Ausführung, wenn Bedingung 1 zutrifft
elif Bedingung 2:
  Ausführung, wenn Bedingung 2 zutrifft, die vorherigen Bedingungen
  jedoch nicht
elif Bedingung 3:
  Ausführung, wenn Bedingung 3 zutrifft, die vorherigen Bedingungen
  jedoch nicht
elif Bedingung N:
  Ausführung, wenn Bedingung N zutrifft, die vorherigen Bedingungen
  jedoch nicht
else:
  Ausführung, wenn keine der Bedingungen zutrifft
```

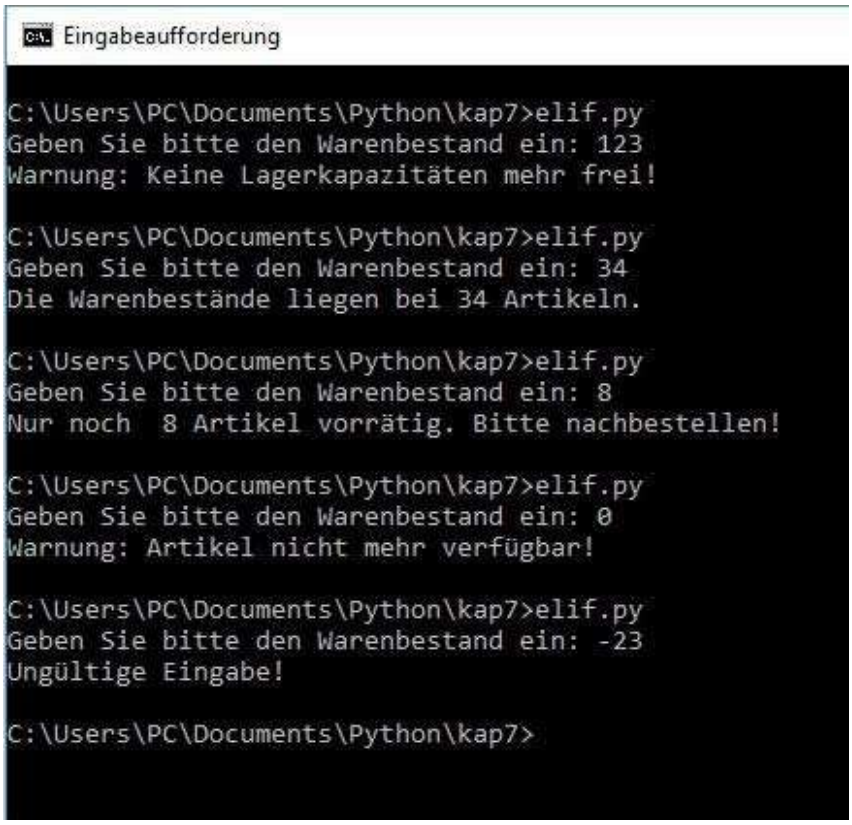
Als Anwendungsbeispiel für eine derartige Konstruktion könnte man sich einen Händler vorstellen, der seine Lagerbestände überprüft. Je nach Warenmenge soll eine bestimmte Aktion durchgeführt werden:

- Erfolgsmeldung, wenn Menge zwischen 10 und 100 liegt
- Warnmeldung, wenn Menge über 100 liegt: keine Lagerkapazitäten mehr vorhanden
- Warnmeldung, wenn der Bestand unter 10, aber mindestens bei 1 liegt: Waren nachbestellen
- Warnmeldung, wenn Bestand bei 0 liegt: Keine Artikel mehr vorhanden
- Fehlermeldung, bei anderen Werten (beispielsweise negative Werte): Ungültige Eingabe

Ein derartiges Programm lässt sich einfach mit mehreren `elif`-Blöcken umsetzen:

```
a = eval(input("Geben Sie bitte den Warenbestand ein: "))
if a >= 10 and a < 100:
   print("Die Warenbestände liegen bei", a,"Artikeln.")
elif a >= 100:
   print("Warnung: Keine Lagerkapazitäten mehr frei!")
elif a > 0:
```

```
    print("Nur noch ", a, "Artikel vorrätig. Bitte nachbestellen!")
elif a == 0:
    print("Warnung: Artikel nicht mehr verfügbar!")
else:
    print("Ungültige Eingabe!")
```



Screenshot 40 Die Ausführung des Programms mit unterschiedlichen Werten

Für ein besseres Verständnis der Abläufe ist es sinnvoll, sich die dritte Bedingung nochmals genau anzuschauen. Diese lautet `elif a > 0`. Das mag auf den ersten Blick verwirrend erscheinen, da dieser Bereich ausgeführt werden soll, wenn der Bestand zwischen 1 und 9 liegt. Dafür wäre eigentlich folgende Bedingung notwendig: `elif a < 10 and a > 0`. Das ist zwar richtig, in diesem Fall jedoch nicht notwendig. Wenn der eingegebene Wert mindestens 10 beträgt, dann trifft entweder die erste oder die zweite Bedingung zu. Das bedeutet, dass wenn das Programm die dritte Abfrage erreicht, der Wert zwingend kleiner als 10 ist. Daher ist es nur noch notwendig, zu überprüfen, ob er größer als 0 ist. Auf diese Weise lässt sich die Bedingung für diese Option etwas einfacher aufstellen.

Nach diesem Schema ist es auch möglich, das Programm weiter zu optimieren. Dafür ist es notwendig, die Positionen der ersten beiden Bedingungen zu tauschen. Für den Bereich zwischen 10 und 100 muss das Programm dann nur noch überprüfen, ob der Wert mindestens 10 beträgt – da für alle Zahlen größer als 100 bereits die erste Bedingung zutrifft. Die Funktionsweise bleibt dabei identisch. Der Code für dieses Programm sieht wie folgt aus:

```
a = eval(input("Geben Sie bitte den Warenbestand ein: "))
if a >= 100:
   print("Warnung: Keine Lagerkapazitäten mehr frei!")
elif a >= 10:
   print("Die Warenbestände liegen bei", a,"Artikeln.")
elif a > 0:
   print("Nur noch ", a, "Artikel vorrätig. Bitte nachbestellen!")
elif a == 0:
   print("Warnung: Artikel nicht mehr verfügbar!")
else:
   print("Ungültige Eingabe!")
```

7.5 Übungsaufgabe: eigene Abfragen erstellen

1. Erstellen Sie ein Programm für einen Gebrauchtwagenhändler, das eine Liste mit drei verschiedenen Fahrzeugen und deren Eigenschaften enthält (Vgl. Kapitel 6.1). Das Programm soll daraufhin den Anwender nach einem Maximalpreis fragen, den er höchstens für den Autokauf aufwenden will. Das Programm soll nun alle Fahrzeuge mit ihren Eigenschaften ausgeben, deren Preis kleiner oder gleich wie der Maximalpreis ist.

2. Schreiben Sie ein Programm, das dem Anwender fünf einfache Rechenaufgaben stellt. Überprüfen Sie die Eingabe. Wenn das Ergebnis richtig ist, erhält der Anwender einen Punkt. Erstellen Sie nach dem Ende der Aufgaben eine Bewertung:

   Bei 0 Punkten: Dringend Nachhilfe benötigt!

   Bei 1 bis 2 Punkten: Viele Fehler: weitere Übung erforderlich!

   Bei 3 bis 4 Punkten: Gute Leistung!

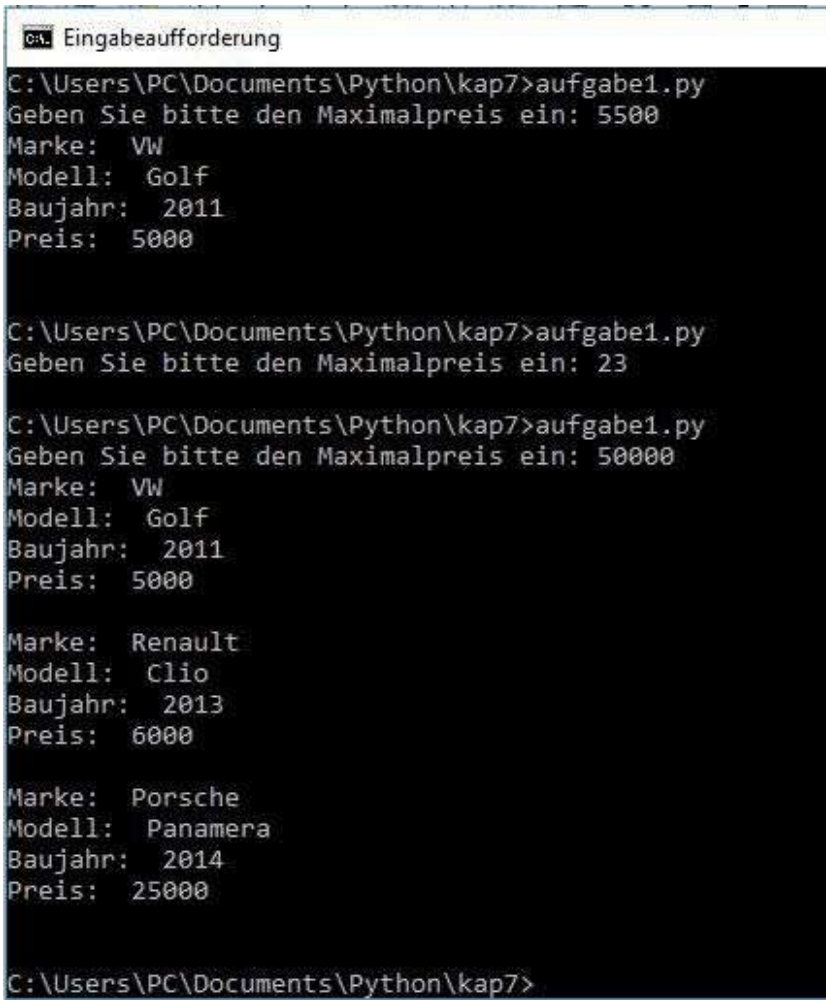
Bei 5 Punkten: Super! Alle Aufgaben richtig gelöst!

Um die richtige Bewertung auszugeben, sollen elif-Blöcke mit möglichst einfachen Bedingungen zum Einsatz kommen.

Lösungen:

1.

```
fahrzeug1 = ["VW", "Golf", 2011, 5000]
fahrzeug2 = ["Renault", "Clio", 2013, 6000]
fahrzeug3 = ["Porsche", "Panamera", 2014, 25000]
listeFahrzeuge = [fahrzeug1, fahrzeug2, fahrzeug3]
maxpreis = eval(input("Geben Sie bitte den Maximalpreis ein: "))
if listeFahrzeuge[0][3] <= maxpreis:
  print ("Marke: ",listeFahrzeuge[0][0])
  print ("Modell: ",listeFahrzeuge[0][1])
  print ("Baujahr: ",listeFahrzeuge[0][2])
  print ("Preis: ",listeFahrzeuge[0][3], "\n")
if listeFahrzeuge[1][3] <= maxpreis:
  print ("Marke: ",listeFahrzeuge[1][0])
  print ("Modell: ",listeFahrzeuge[1][1])
  print ("Baujahr: ",listeFahrzeuge[1][2])
  print ("Preis: ",listeFahrzeuge[1][3], "\n")
if listeFahrzeuge[2][3] <= maxpreis:
  print ("Marke: ",listeFahrzeuge[2][0])
  print ("Modell: ",listeFahrzeuge[2][1])
  print ("Baujahr: ",listeFahrzeuge[2][2])
  print ("Preis: ",listeFahrzeuge[2][3], "\n")
```



Screenshot 41 Das Verhalten des Programms bei unterschiedlichen Eingaben

2.

```
punkte = 0
ergebnis = eval(input("Aufgabe 1: Was ist das Ergebnis aus 6 + 7? "))
if ergebnis == 13:
  punkte = punkte + 1
ergebnis = eval(input("Aufgabe 2: Was ist das Ergebnis aus 12 - 4?
"))
if ergebnis == 8:
  punkte = punkte + 1
ergebnis = eval(input("Aufgabe 3: Was ist das Ergebnis aus 4 * 5? "))
if ergebnis == 20:
  punkte = punkte + 1
ergebnis = eval(input("Aufgabe 4: Was ist das Ergebnis aus 24 / 6?
"))
```

```
if ergebnis == 4:
  punkte = punkte + 1
ergebnis = eval(input("Aufgabe 5: Was ist das Ergebnis aus 5 + 9 +
24? "))
if ergebnis == 38:
  punkte = punkte + 1
if punkte == 0:
  print ("Dringend Nachhilfe benötigt!")
elif punkte < 3:
  print ("Viele Fehler: weitere Übung erforderlich!")
elif punkte < 5:
  print ("Gute Leistung!")
else:
  print ("Super! Alle Aufgaben richtig gelöst!")
```

Screenshot 42 Die Bewertung orientiert sich am erreichten Punktestand.