1.1 Variablen, Eingabe und Ausgabe

Leitidee

Ein Programm ist eine Liste von Anweisungen, die der Computer der Reihe nach ausführt – von oben nach unten. In der Blockprogrammierung hast du die Anweisungen als Bausteine zusammengeschoben. In Python schreibst du sie als Text. In diesem Abschnitt lernst du die drei wichtigsten Dinge, die fast jedes Programm braucht: Es **gibt etwas aus**, es **merkt sich Werte** und es **liest Eingaben** der Nutzerin oder des Nutzers ein. Am Ende schreibst du ein Programm, das dich nach deinem Namen fragt und dir einen Steckbrief ausgibt.

Lernziele

Nach diesem Abschnitt kannst du

- mit `print()` Texte und Zahlen auf dem Bildschirm ausgeben,
- Variablen anlegen, ihnen Werte zuweisen und mit ihnen rechnen,
- die Datentypen `int`, `float`, `str` und `bool` unterscheiden,
- mit `input()` Eingaben einlesen und mit `int()` und `float()` in Zahlen umwandeln,
- eine Fehlermeldung lesen und die Ursache des Fehlers finden.

1.1.1 Das erste Programm: Ausgaben mit `print`

Das kürzeste sinnvolle Programm besteht aus einer einzigen Zeile:

```
print("Hallo Welt!")
```

AUSGABE

```
Hallo Welt!
```

Das Wort `print` ist englisch und heißt „drucken“. Gemeint ist: Zeige etwas auf dem Bildschirm an. Was angezeigt werden soll, steht in den **runden Klammern** dahinter. Der Text steht in **Anführungszeichen**. Die Anführungszeichen selbst erscheinen nicht in der Ausgabe. Sie zeigen Python nur, wo der Text anfängt und wo er aufhört.

Ein Programm hat meistens mehrere Zeilen. Python führt sie **nacheinander von oben nach unten** aus. Jede `print`-Anweisung beginnt eine neue Zeile in der Ausgabe:

```
1 print("Guten Morgen!")
2 print("Heute lernen wir Python.")
3 print("Los geht's.")
```

AUSGABE

```
Guten Morgen!
Heute lernen wir Python.
Los geht's.
```

Text oder Zahl?

Mit `print` kannst du auch Zahlen ausgeben und sogar rechnen. Vergleiche die beiden Zeilen genau:

```
print(3 + 4)
print("3 + 4")
```

AUSGABE

```
7
3 + 4
```

- **Zeile 1:** Hier stehen keine Anführungszeichen. Python liest `3 + 4` als Rechnung, rechnet sie aus und gibt das Ergebnis `7` aus.
- **Zeile 2:** Jetzt steht `3 + 4` in Anführungszeichen. Für Python ist das ein Text wie jeder andere. Er wird genau so ausgegeben, wie er dasteht.

Du kannst in eine `print`-Anweisung auch mehrere Dinge schreiben. Trenne sie durch **Kommas**. Python setzt in der Ausgabe automatisch je ein Leerzeichen dazwischen:

```
print("Ich bin", 15, "Jahre alt.")
print("In fünf Jahren bin ich", 15 + 5, "Jahre alt.")
```

AUSGABE

```
Ich bin 15 Jahre alt.
In fünf Jahren bin ich 20 Jahre alt.
```

Merke

Mit `print(...)` gibt man Werte auf dem Bildschirm aus. Texte stehen in Anführungszeichen, Zahlen und Rechnungen ohne. Mehrere Werte trennt man durch Kommas.

Versuch: Probiere es aus

Tippe das Programm mit „Guten Morgen!“ ab und starte es. Ändere dann die Texte. Was passiert, wenn du eine der drei Zeilen an den Anfang verschiebst?

Wenn etwas schiefgeht

Beim Abtippen passieren Fehler – auch Profis jeden Tag. Python bricht dann ab und zeigt eine **Fehlermeldung**. Das ist kein Grund zur Sorge: Die Meldung ist ein Hinweis, wo das Problem liegt. Hier fehlt das schließende Anführungszeichen:

```
print("Hallo)
```

FEHLERMELDUNG

```
SyntaxError: unterminated string literal (detected at line 1)
```

Die Fehlermeldung ist englisch. **SyntaxError** bedeutet: Die Schreibweise stimmt nicht – Python versteht den Satz nicht. *Unterminated string literal* heißt „nicht abgeschlossener Text“. Python sagt dir sogar die Zeile: *detected at line 1*. Die Lösung: das fehlende `"` ergänzen.

Und was passiert, wenn du die Anführungszeichen ganz vergisst?

```
print(Hallo)
```

FEHLERMELDUNG

```
NameError: name 'Hallo' is not defined
```

Ohne Anführungszeichen hält Python `Hallo` nicht für einen Text, sondern für einen **Namen** – so wie die Variablen, die du gleich kennenlernst. Einen Namen `Hallo` gibt es aber nicht. Deshalb meldet Python einen **NameError**: „Der Name ‘Hallo’ ist nicht definiert.“

Stolperstein

Die häufigsten Tippfehler am Anfang: ein vergessenes Anführungszeichen, eine vergessene Klammer, `Print` statt `print` (Python unterscheidet Groß- und Kleinschreibung!) und deutsche Anführungszeichen „…“ statt `"…"`. Wenn eine Fehlermeldung kommt, schau dir zuerst die genannte Zeile an – und die Zeile davor.

1.1.2 Variablen: Werte speichern

Programme müssen sich Dinge merken: einen Namen, einen Punktestand, einen Preis. Dafür gibt es **Variablen**.

Stell dir eine Variable wie eine **beschriftete Schachtel** vor. Außen auf der Schachtel steht ein Name, zum Beispiel `alter`. In der Schachtel liegt ein Wert, zum Beispiel `15`. Immer wenn das Programm den Namen `alter` liest, schaut es in die Schachtel und nimmt den Wert, der gerade darin liegt.



Drei Variablen: Außen steht der Name, innen liegt der Wert. Darunter steht, welche Art von Wert es ist (der Datentyp)

Definition: Variable

Eine **Variable** ist ein benannter Speicherplatz im Arbeitsspeicher des Computers, in dem ein Wert abgelegt wird. Über ihren Namen kann das Programm den Wert lesen und verändern.

In Python legst du eine Variable an, indem du ihr einen Wert gibst:

```
1 alter = 15
2 print(alter)
3 print("Ich bin", alter, "Jahre alt.")
```

AUSGABE

```
15
Ich bin 15 Jahre alt.
```

- **Zeile 1:** Die Variable `alter` wird angelegt und bekommt den Wert `15`. Auf dem Bildschirm passiert dabei nichts.
- **Zeile 2:** `alter` steht ohne Anführungszeichen. Python schaut deshalb in die Schachtel `alter` und gibt ihren Inhalt aus: `15`.
- **Zeile 3:** Text und Variable werden durch Kommas getrennt ausgegeben.

Das Gleichheitszeichen ist ein Zuweisungszeichen

Das Zeichen = bedeutet in Python **nicht** „ist gleich“ wie in Mathe. Es bedeutet: **„bekommt den Wert“**. Lies `alter = 15` also so: „Die Variable `alter` bekommt den Wert 15.“ Man nennt das eine **Zuweisung**. Dabei wird immer zuerst die rechte Seite ausgerechnet, dann wird das Ergebnis in die Variable links gelegt.

Deshalb ist die folgende Zeile in Python sinnvoll, obwohl sie in Mathe falsch wäre:

```
1 alter = 15
2 print(alter)
3 alter = alter + 1
4 print(alter)
```

AUSGABE

```
15
16
```

- **Zeile 1:** `alter` bekommt den Wert `15`.
- **Zeile 3:** Zuerst wird die rechte Seite ausgerechnet: Der aktuelle Wert von `alter` ist `15`, plus `1` ergibt `16`. Dann bekommt `alter` diesen neuen Wert. Der alte Wert `15` ist weg – er wurde **überschrieben**.

Wie sich die Werte Schritt für Schritt ändern, zeigt ein **Ablaufprotokoll**. Darin steht nach jeder ausgeführten Zeile, welchen Wert die Variable gerade hat:

Schritt	Zeile	ausgeführte Anweisung	`alter` danach
1	1	`alter = 15`	15
2	2	`print(alter)`	15
3	3	`alter = alter + 1`	16
4	4	`print(alter)`	16

Gymnasium Ottweiler

Merke

- Eine **Variable** ist ein benannter Speicherplatz im Arbeitsspeicher, in dem ein Wert abgelegt wird.
- Mit dem **Zuweisungsoperator** = wird einer Variablen ein Wert zugewiesen, z. B. `alter = 15`. Zuerst wird die rechte Seite ausgewertet, dann wird das Ergebnis in der Variablen links gespeichert.

Gute Namen für Variablen

Den Namen einer Variablen darfst du selbst wählen. Dabei gelten ein paar Regeln:

- Erlaubt sind Buchstaben, Ziffern und der Unterstrich _. Ein Name darf **nicht mit einer Ziffer beginnen** und **keine Leerzeichen** enthalten.
- Groß- und Kleinschreibung zählt: `alter` und `Alter` sind zwei verschiedene Namen.
- Verwende **sprechende Namen**, an denen man erkennt, was gespeichert ist: `preis` statt `p`, `anzahl_artikel` statt `aa`. Mehrere Wörter verbindest du mit _.
- Verzichte auf Umlaute und ß in Namen: `groesse` statt `größe`. Das funktioniert zwar meist, kann aber Probleme machen.

Name	erlaubt?	Grund
`punkte`	✓	
`punkte_runde_2`	✓	
`2_runde`	✗	beginnt mit einer Ziffer
`meine punkte`	✗	enthält ein Leerzeichen

Stolperstein

Wenn du eine Variable benutzt, bevor sie einen Wert bekommen hat, oder dich beim Namen vertippst, meldet Python einen **NameError**:

```
alter = 15
print(Alter)
```

FEHLERMELDUNG

```
NameError: name 'Alter' is not defined. Did you mean: 'alter'?
```

Neuere Python-Versionen geben dir sogar einen Tipp: *Did you mean: 'alter'?* – „Meintest du 'alter'?"

Arbeitsauftrag 1: Vorhersage

Lies das Programm, **ohne** es auszuführen. Schreibe auf, was es ausgibt. Überprüfe deine Vorhersage danach am Computer.

```
1 punkte = 10
2 print(punkte)
3 punkte = punkte + 5
4 punkte = punkte * 2
5 print("Endstand:", punkte)
```

Meine Vorhersage:

1.1.3 Datentypen: Zahlen, Texte und Wahrheitswerte

In einer Variablen können ganz unterschiedliche Arten von Werten liegen: ganze Zahlen, Kommazahlen, Texte. Die Art eines Wertes heißt **Datentyp**. Python merkt sich zu jedem Wert seinen Datentyp, denn davon hängt ab, was man mit dem Wert machen kann. Mit Zahlen kann man rechnen, mit Texten nicht.

Datentyp	Bedeutung	Beispiele
`int`	ganze Zahl (englisch *integer*)	`15`, `-3`, `0`, `2026`
`float`	Kommazahl (englisch *floating point number*)	`1.68`, `-0.5`, `3.0`
`str`	Zeichenkette, also Text (englisch *string*)	`"Alex"`, `"Hallo Welt!"`, `"15"`
`bool`	Wahrheitswert (nach George Boole)	`True` (wahr), `False` (falsch)

Definition: Datentyp

Der **Datentyp** eines Wertes legt fest, welche Art von Daten er darstellt (z. B. ganze Zahl, Kommazahl, Text oder Wahrheitswert) und welche Operationen damit möglich sind.

Drei Dinge sind besonders wichtig:

- Kommazahlen schreibt man in Python mit einem **Punkt**, nicht mit einem Komma: `1.68`, nicht `1,68`.
- Alles in Anführungszeichen ist ein Text, auch wenn Ziffern darin stehen: `"15"` ist eine Zeichenkette, `15` eine Zahl.
- `True` und `False` schreibt man mit großem Anfangsbuchstaben und ohne Anführungszeichen. Wahrheitswerte brauchst du später für Bedingungen.

Mit `type(...)` kannst du Python fragen, welchen Datentyp ein Wert hat:

```
1 name = "Alex"
2 alter = 15
3 groesse = 1.68
4 volljaehrig = False
5 print(type(name))
6 print(type(alter))
7 print(type(groesse))
8 print(type(volljaehrig))
```

AUSGABE
```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

Die Ausgabe `<class 'str'>` bedeutet einfach: „Dieser Wert ist vom Typ `str`."

Merke

Python unterscheidet u. a. die Datentypen `int` (Ganzzahl), `float` (Kommazahl), `str` (Zeichenkette) und `bool` (Wahrheitswert).

Rechnen mit Zahlen

Mit `int`- und `float`-Werten kannst du rechnen. Die Grundrechenarten schreibt man mit `+`, `-`, `*` (mal) und `/` (geteilt). Es gilt wie in Mathe **Punkt vor Strich**, und Klammern werden zuerst ausgerechnet:

```
1 a = 7
2 b = 2
3 print(a + b)
4 print(a - b)
5 print(a * b)
6 print(a / b)
7 print(2 + 3 * 4)
8 print((2 + 3) * 4)
```

AUSGABE
```
9
5
14
3.5
14
20
```

Beachte Zeile 6: Das Ergebnis einer Division mit `/` ist **immer** ein `float`, auch wenn es aufgeht. `6 / 2` ergibt `3.0`, nicht `3`. Weitere Rechenzeichen lernst du im nächsten Abschnitt kennen.

Stolperstein

Schreibst du eine Kommazahl versehentlich mit Komma, gibt es **keine** Fehlermeldung – Python versteht aber etwas völlig anderes:

```
preis = 2,50
print(preis)
```

AUSGABE
```
(2, 50)
```

Python hat zwei Zahlen gespeichert, `2` und `50`. Mit `preis` kann man jetzt nicht mehr rechnen. Richtig ist `preis = 2.50`.

Arbeitsauftrag 2: Datentypen bestimmen

Gib zu jedem Wert den Datentyp an (`int`, `float`, `str` oder `bool`). Überprüfe deine Antworten anschließend mit `type(...)`.

Wert	42	"42"	4.2	True	"True"	-7	7 / 7	"Hallo"
Datentyp								

1.1.4 Eingaben mit `input`

Bisher stehen alle Werte fest im Programm. Spannender wird es, wenn die Nutzerin oder der Nutzer etwas eingeben kann. Dafür gibt es `input(...)`:

```
name = input("Wie heißt du? ")
print("Hallo", name)
```

AUSGABE

```
Wie heißt du? Lena
Hallo Lena
```

- **Zeile 1:** `input` zeigt zuerst den Text in den Klammern an, die sogenannte **Eingabeaufforderung**. Dann **wartet das Programm**, bis jemand etwas eintippt und die Eingabetaste drückt. Der eingetippte Text (hier blau gedruckt) wird in der Variablen `name` gespeichert.
- **Zeile 2:** Der gespeicherte Name wird ausgegeben.

Tipp

Setze ans Ende der Eingabeaufforderung ein Leerzeichen: `"Wie heißt du? "`. Sonst klebt die Eingabe direkt am Fragezeichen.

Die Eingabe ist immer ein Text

Probiere einmal dieses Programm mit der Eingabe `7`. Was erwartest du?

```
1 zahl = input("Gib eine Zahl ein: ")
2 print(zahl + zahl)
3 print(type(zahl))
```

AUSGABE

```
Gib eine Zahl ein: 7
77
<class 'str'>
```

Überraschung: Python rechnet nicht `7 + 7 = 14`, sondern gibt `77` aus. Der Grund steht in der letzten Zeile: `zahl` hat den Typ `str`. Das, was jemand eintippt, ist für Python **immer eine Zeichenkette**, auch wenn nur Ziffern darin stehen. Und das Pluszeichen hängt zwei Zeichenketten einfach aneinander: `"7" + "7"` ergibt `"77"`.

Merke

- Mit `input(...)` liest man eine Eingabe von der Nutzerin oder dem Nutzer ein. **Die Eingabe ist immer vom Typ `str`!**

1.1.5 Typumwandlung: aus Text wird eine Zahl

Damit du mit einer Eingabe rechnen kannst, musst du sie in eine Zahl **umwandeln**. Dafür gibt es drei Befehle, die genauso heißen wie die Datentypen:

Befehl	wandelt um in	Beispiel	Ergebnis
`int(...)`	ganze Zahl	`int("7")`	`7`
`float(...)`	Kommazahl	`float("2.5")`	`2.5`
`str(...)`	Zeichenkette	`str(15)`	`"15"`

Definition: Typumwandlung

Bei einer **Typumwandlung** wird ein Wert in einen anderen Datentyp umgewandelt, zum Beispiel die Zeichenkette `"7"` in die ganze Zahl `7`.

Das folgende Programm zeigt alles zusammen. Es fragt nach einer Lieblingszahl, zeigt, dass die Eingabe ein Text ist, und rechnet nach der Umwandlung mit ihr:

```
1 lieblingszahl = input("Gib deine Lieblingszahl ein: ")
2 print("Deine Eingabe war:", lieblingszahl, "- Typ:", type(lieblingszahl))
3
4 lieblingszahl_int = int(lieblingszahl)
5 print("Deine Zahl mal 2 ist:", lieblingszahl_int * 2)
```

AUSGABE

```
Gib deine Lieblingszahl ein: 7
Deine Eingabe war: 7 - Typ: <class 'str'>
Deine Zahl mal 2 ist: 14
```

- **Zeile 1:** Die Eingabe wird als Zeichenkette in `lieblingszahl` gespeichert.
- **Zeile 2:** Die Ausgabe bestätigt: Der Typ ist `str`.
- **Zeile 3:** Eine leere Zeile. Sie hat keine Wirkung, macht das Programm aber übersichtlicher.
- **Zeile 4:** `int(lieblingszahl)` macht aus dem Text `"7"` die Zahl `7`. Das Ergebnis wird in einer neuen Variablen gespeichert.
- **Zeile 5:** Jetzt wird wirklich gerechnet: `7 * 2 = 14`.

Eingabe und Umwandlung in einer Zeile

Meistens schreibt man Eingabe und Umwandlung in **eine** Zeile. Das ist kürzer, und man braucht keine zweite Variable:

```
alter = int(input("Wie alt bist du? "))
print("Nächstes Jahr bist du", alter + 1)
```

AUSGABE

```
Wie alt bist du? 15
Nächstes Jahr bist du 16
```

Lies Zeile 1 **von innen nach außen**: Zuerst wird `input(...)` ausgeführt und liefert den Text `"15"`. Dieser Text wird an `int(...)` übergeben und in die Zahl `15` umgewandelt. Erst diese Zahl wird in `alter` gespeichert. Achte darauf, dass am Ende **zwei** schließende Klammern stehen – eine für `input`, eine für `int`.

Für Kommazahlen nimmst du `float` statt `int`:

```
groesse = float(input("Wie groß bist du in Metern? "))
print("In Zentimetern sind das", groesse * 100)
```

AUSGABE

```
Wie groß bist du in Metern? 1.68
In Zentimetern sind das 168.0
```

Merke

Soll eine Eingabe als Zahl weiterverarbeitet werden, muss sie umgewandelt werden: `int(...)` bzw. `float(...)` (**Typumwandlung**).

Stolperstein

Die Umwandlung klappt nur, wenn der Text wirklich wie eine Zahl aussieht. Sonst meldet Python einen **ValueError** („falscher Wert“):

```
alter = int(input("Wie alt bist du? "))
```

FEHLERMELDUNG

```
ValueError: invalid literal for int() with base 10: 'sieben'
```

Dasselbe passiert bei `int("2.5")` (eine Kommazahl ist keine ganze Zahl) und bei `float("2,5")` (Komma statt Punkt). Bei einem ValueError lohnt sich also ein Blick darauf, **was eingegeben wurde**.

1.1.6 Texte zusammensetzen

Mit Kommas in `print` setzt Python immer ein Leerzeichen zwischen die Teile. Manchmal willst du das nicht, zum Beispiel wenn direkt nach einem Namen ein Komma oder ein Punkt folgen soll. Dann kannst du Texte mit `+` **aneinanderhängen** (verketten). Dabei kommt kein Leerzeichen dazu – die brauchst du selbst in den Anführungszeichen:

```
1 name = "Alex"
2 print("Hallo, " + name + "!")
3 print("Hallo,", name, "!")
```

AUSGABE

```
Hallo, Alex!
Hallo, Alex !
```

In Zeile 3 stört das Leerzeichen vor dem Ausrufezeichen. Mit `+` (Zeile 2) sitzt alles genau richtig.

Stolperstein

Mit `+` lassen sich nur **Texte mit Texten** verbinden. Eine Zahl muss vorher mit `str(...)` in einen Text umgewandelt werden. Sonst bricht das Programm ab:

```
alter = 15
print("Du bist " + alter + " Jahre alt.")
```

FEHLERMELDUNG

```
Traceback (most recent call last):
  File "steckbrief.py", line 2, in <module>
    print("Du bist " + alter + " Jahre alt.")
          ~~~~~~~~~~^~~~~~~
TypeError: can only concatenate str (not "int") to str
```

Richtig ist: `print("Du bist " + str(alter) + " Jahre alt.")`

Eine Fehlermeldung lesen

Die Meldung oben ist eine vollständige Fehlermeldung, wie du sie auf dem Bildschirm siehst. Lies sie **von unten nach oben**:

1. **Letzte Zeile:** die Art des Fehlers und eine kurze Beschreibung. Hier: `TypeError` – ein Fehler mit den Datentypen. *can only concatenate str (not "int") to str* heißt: „Kann nur `str` (nicht `int`) an `str` anhängen."
2. **Die Zeilen darüber** zeigen, wo der Fehler passiert ist: in Zeile 2 (*line 2*). Darunter steht diese Zeile noch einmal, und die Zeichen ~~~^~~~ zeigen auf die Stelle, an der der Fehler auftritt.
3. Die erste Zeile *Traceback (most recent call last)* kannst du überlesen.

Die vier Fehlerarten, die dir am Anfang am häufigsten begegnen:

Fehler	Bedeutung	typische Ursache
`SyntaxError`	Schreibweise falsch	Anführungszeichen oder Klammer fehlt
`NameError`	Name unbekannt	Tippfehler im Namen, Variable noch nicht angelegt, Anführungszeichen vergessen
`TypeError`	Datentyp passt nicht	Zahl und Text mit `+` verbunden
`ValueError`	Wert passt nicht	`int("sieben")`, `float("2,5")`

So kannst du es sagen

So beschreibst du ein Programm in Fachsprache – vom einfachen zum genauen Satz:

- *„alter ist 15."* → *„Der Variablen `alter` wird der Wert 15 zugewiesen."*
- *„Das Programm fragt nach dem Namen."* → *„Die Eingabeanweisung liest den Namen ein und speichert ihn als Zeichenkette in der Variablen `name`."*
- *„Da kommt 14 raus."* → *„Zur Eingabe 7 gibt das Programm den Wert 14 aus."*

1.1.7 Übungen

Arbeitsauftrag 3: Fehler finden

Das folgende Programm soll nach zwei Zahlen fragen und ihre Summe ausgeben. Es enthält aber **drei Fehler**. Finde sie, erkläre jeweils, welche Fehlermeldung oder welches falsche Verhalten sie auslösen, und schreibe das Programm richtig auf.

```
1 zahl1 = input("Erste Zahl: )
2 zahl2 = input("Zweite Zahl: ")
3 print("Die Summe ist", Zahl1 + zahl2)
```

Die Fehler:

Arbeitsauftrag 4: Steckbrief

Schreibe ein Programm, das folgende Angaben mit `input()` abfragt und anschließend als vollständigen Satz ausgibt:

- Name
- Alter (als Zahl!)
- Lieblingsfach

Gib zum Schluss in einer zweiten Zeile das **doppelte Alter** aus. Daran siehst du, ob das Alter wirklich als Zahl gespeichert ist.

So könnte die Ausgabe aussehen:

AUSGABE

```
Wie heißt du? Alex
Wie alt bist du? 15
Was ist dein Lieblingsfach? Informatik
Hallo, ich bin Alex, 15 Jahre alt und mein Lieblingsfach ist Informatik.
Doppeltes Alter: 30
```

Arbeitsauftrag 5: Einfacher Rechner

Schreibe ein Programm, das zwei Zahlen einliest (denk an die Typumwandlung!) und **Summe, Differenz und Produkt** der beiden Zahlen ausgibt.

Werkzeug

Mit `round(wert, stellen)` rundest du eine Zahl auf eine bestimmte Anzahl von Nachkommastellen. `round(3.14159, 2)` ergibt `3.14`. Lässt du die Stellen weg, wird auf eine ganze Zahl gerundet: `round(2.7)` ergibt `3`.

Arbeitsauftrag 6 (Zusatz): Gesamtpreis

Frage einen Preis in Euro (als Kommazahl, `float`) sowie eine Anzahl an Artikeln (als `int`) ab und berechne den Gesamtpreis. Runde die Ausgabe sinnvoll (Tipp: `round(wert, 2)`).

1.1.8 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 7: Mit oder ohne Anführungszeichen? ●○○

Lies das Programm, **ohne** es auszuführen. Schreibe auf, was es ausgibt. Überprüfe deine Vorhersage danach am Computer.

```
1 name = "Mia"
2 alter = 14
3 print("Name:", name)
4 print("alter")
5 print(alter)
6 print("In drei Jahren bist du", alter + 3)
```

Meine Vorhersage:

Gymnasium Ottweiler

Arbeitsauftrag 8: Welcher Datentyp? ●○○

Das Programm wird mit den Eingaben `12`, `12` und `1.5` ausgeführt. Gib zu jeder Variablen an, welchen Datentyp ihr Wert hat (`int`, `float`, `str` oder `bool`). Überprüfe deine Antworten anschließend am Computer mit `print(type(...))`.

```
1 a = input("Gib eine Zahl ein: ")
2 b = int(input("Gib eine Zahl ein: "))
3 c = float(input("Gib eine Zahl ein: "))
4 d = "3.5"
5 e = 3.5
6 f = False
```

Variable	a	b	c	d	e	f
Datentyp						

Arbeitsauftrag 9: Kontostand verfolgen ●○○

Fülle das Ablaufprotokoll aus. Trage nach jedem Schritt die Werte der beiden Variablen ein (– bedeutet: noch kein Wert). Gib dann an, was das Programm ausgibt.

```
1 kontostand = 20
2 kontostand = kontostand - 5
3 taschengeld = 15
4 kontostand = kontostand + taschengeld
5 print("Kontostand:", kontostand)
```

Schritt	Zeile	`kontostand`	`taschengeld`
1	1		
2	2		
3	3		
4	4		
5	5		

Ausgabe:

Arbeitsauftrag 10: Handyvertrag ●○○

Das Programm soll den monatlichen Preis eines Handyvertrags (Kommazahl) und die Laufzeit in Monaten (ganze Zahl) einlesen und die Gesamtkosten ausgeben, auf Cent gerundet. Ergänze die drei Lücken `___` und teste dein Programm mit den Eingaben `9.99` und `24`.

```
1 preis = ___(input("Preis pro Monat in Euro: "))
2 monate = ___(input("Laufzeit in Monaten: "))
3 gesamt = preis * monate
4 print("Gesamtkosten:", ___, "Euro")
```

Arbeitsauftrag 11: Fehler finden: Temperatur umrechnen ●●○

In den USA misst man Temperaturen in Grad Fahrenheit. Man rechnet so um: Grad Fahrenheit = Grad Celsius · 1,8 + 32. Das Programm soll eine Temperatur in Grad Celsius einlesen und in Grad Fahrenheit umrechnen. Es enthält **drei Fehler**.

```
1 celsius = input("Temperatur in Grad Celsius: ")
2 fahrenheit = celsius * 1.8 + 32
3 print("Das sind", Fahrenheit, "Grad Fahrenheit.)
```

1. Finde die drei Fehler. Notiere jeweils die Zeile und was falsch ist.
2. Schreibe das Programm richtig auf und teste es mit `20` (richtig ist 68 Grad Fahrenheit).

Die Fehler:

Arbeitsauftrag 12: Taschengeld ●●○

Schreibe ein Programm, das das wöchentliche Taschengeld (Kommazahl) und eine Anzahl von Wochen (ganze Zahl) einliest. Es soll ausgeben, wie viel Geld in dieser Zeit zusammenkommt. Danach soll es den Preis eines Wunschartikels einlesen und ausgeben, wie viel Geld nach dem Kauf noch übrig ist. Runde die Beträge auf Cent.

So könnte es aussehen:

AUSGABE

```
Taschengeld pro Woche in Euro: 7.5
Anzahl der Wochen: 6
In 6 Wochen bekommst du 45.0 Euro.
Preis des Wunschartikels in Euro: 29.99
Nach dem Kauf bleiben 15.01 Euro übrig.
```

Arbeitsauftrag 13: Pfannkuchen für alle ●●○

Für **eine** Person braucht man für Pfannkuchen 62,5 g Mehl, 0,5 Eier und 125 ml Milch.

1. Schreibe ein Programm, das die Anzahl der Personen einliest und ausgibt, wie viel man von jeder Zutat braucht.
2. Begründe, warum du die Anzahl der Personen mit `int` und nicht mit `float` umwandelst.

Begründung:

Arbeitsauftrag 14: Busfahrt oder Monatskarte? ●●●

Eine Einzelfahrt mit dem Bus kostet 2,10 Euro. Eine Monatskarte kostet 58 Euro.

1. Schreibe ein Programm, das einliest, wie viele Busfahrten man an einem Schultag macht und an wie vielen Schultagen man im Monat fährt. Es soll ausgeben, was die Einzelfahrten im Monat zusammen kosten. Danach soll es ausgeben, wie viel man mit der Monatskarte spart, also die Kosten der Einzelfahrten minus 58 Euro.
2. Teste dein Programm mit `2` Fahrten an `20` Tagen und mit `2` Fahrten an `10` Tagen. Erkläre, was ein **negatives** Ergebnis für die Ersparnis bedeutet.

Testergebnisse und Erklärung:

Gymnasium Ottweiler

1.1.9 Das Wichtigste in Kürze

- Ein Programm ist eine Folge von Anweisungen, die von oben nach unten ausgeführt werden.
- `print(...)` gibt Werte aus. Mehrere Werte werden durch Kommas getrennt und mit Leerzeichen ausgegeben.
- Eine **Variable** ist ein benannter Speicherplatz. Mit = bekommt sie einen Wert (**Zuweisung**); ein neuer Wert überschreibt den alten.
- Wichtige **Datentypen**: `int` (ganze Zahl), `float` (Kommazahl mit Punkt), `str` (Zeichenkette) und `bool` (`True`/`False`).
- `input(...)` liest eine Eingabe ein. Die Eingabe ist **immer** vom Typ `str`.
- Mit `int(...)`, `float(...)` und `str(...)` wandelt man Werte in einen anderen Datentyp um (**Typumwandlung**).
- Fehlermeldungen liest man von unten nach oben: Fehlerart, Beschreibung, dann die Zeilennummer.

Selbstcheck

- ☐ Ich kann Texte und Zahlen mit `print` ausgeben.
- ☐ Ich kann erklären, was eine Variable ist und was bei `x = x + 1` passiert.
- ☐ Ich kenne die Datentypen `int`, `float`, `str` und `bool` und kann zu einem Wert den Typ angeben.
- ☐ Ich kann mit `input` eine Eingabe einlesen und weiß, dass sie immer eine Zeichenkette ist.
- ☐ Ich kann eine Eingabe mit `int` oder `float` in eine Zahl umwandeln.
- ☐ Ich kann eine Fehlermeldung lesen und weiß, was `SyntaxError`, `NameError`, `TypeError` und `ValueError` bedeuten.