1. (4 Punkte) Gegeben ist der folgende Python-Code:

```
preis = 4.50                    # Zeile 1
anzahl = 3                      # Zeile 2
anzahl = anzahl + 2             # Zeile 3
gesamt = preis * anzahl         # Zeile 4
```

   (a) Gib alle Bezeichner an.

   (b) Gib für jede Zeile an, ob es sich um eine Initialisierung oder um eine (erneute) Wertzuweisung handelt.

   (c) Gib nach Zeile 4 für jede Variable ihren Wert und ihren Datentyp an.

   (d) Erläutere, warum Zeile 3 sinnvoll ist, obwohl sie als mathematische Gleichung keinen Sinn ergibt.

Lösung:

(a) `preis`, `anzahl`, `gesamt`

(b) Zeile 1, 2 und 4: Initialisierung (erste Wertzuweisung). Zeile 3: erneute Wertzuweisung an `anzahl`.

(c) `preis`: 4.5 (`float`), `anzahl`: 5 (`int`), `gesamt`: 22.5 (`float`)

(d) Eine Wertzuweisung wird von rechts nach links ausgeführt: Zuerst wird `anzahl + 2` mit dem alten Wert 3 ausgewertet, dann wird das Ergebnis 5 der Variable `anzahl` zugewiesen. Der alte Wert geht dabei verloren.

2. (4 Punkte) Gegeben ist der folgende Java-Code:

```
int punkte;                     // Zeile 1
punkte = 10;                    // Zeile 2
double schnitt = 2.5;           // Zeile 3
punkte = punkte * 2;            // Zeile 4
punkte = "zwanzig";             // Zeile 5
```

   (a) Gib für die Zeilen 1 bis 4 an, ob eine Deklaration, eine Initialisierung oder eine Wertzuweisung vorliegt (auch mehreres ist möglich).

   (b) Zeile 5 führt zu einem Fehler. Erläutere, warum.

   (c) In Python wäre die Wertzuweisung `punkte = 'zwanzig'` erlaubt. Erläutere den Unterschied zwischen Java und Python.

Lösung:

(a) Zeile 1: Deklaration (Bezeichner `punkte`, Datentyp `int`). Zeile 2: Initialisierung. Zeile 3: Deklaration und Initialisierung in einem Schritt. Zeile 4: Wertzuweisung.

(b) `punkte` ist als `int` deklariert. In Java ist der Datentyp einer Variable fest, ein String passt nicht in eine `int`-Variable. Der Fehler wird schon beim Übersetzen gemeldet.

(c) In Java gehört der Datentyp zur Variable und wird bei der Deklaration festgelegt (statische Typisierung). In Python gibt es keine eigene Deklaration, die Variable entsteht bei der ersten Wertzuweisung. Der Datentyp gehört zum Wert, deshalb kann eine Variable später einen Wert eines anderen Typs bekommen (dynamische Typisierung).

3. (4 Punkte) Gegeben ist der folgende Python-Code:

```
bonus = 5

def punkte_berechnen(punkte):
    gesamt = punkte + bonus
    return gesamt

ergebnis = punkte_berechnen(20)
print(ergebnis)
print(bonus)
print(gesamt)
```

   (a) Gib alle globalen und alle lokalen Variablen an.

   (b) Gib an, was die ersten beiden `print`-Anweisungen ausgeben.

   (c) Die letzte Zeile führt zu einem Fehler. Erläutere mit dem Begriff Gültigkeitsbereich, warum.

Lösung:

(a) Global: `bonus`, `ergebnis`. Lokal (in der Funktion): `punkte`, `gesamt`.

(b) `25` und `5`

(c) `gesamt` ist eine lokale Variable der Funktion. Ihr Gültigkeitsbereich ist nur die Funktion, und sie existiert nur, solange die Funktion ausgeführt wird. Außerhalb ist sie nicht bekannt, deshalb gibt es einen `NameError`.

4. (3 Punkte) Gegeben ist der folgende Python-Code:

```
x = 1

def f():
    x = 2
    print(x)

f()
print(x)
```

   (a) Gib an, was das Programm ausgibt.

   (b) Erläutere die Ausgabe.

Lösung:

(a) `2` und danach `1`

(b) Durch die Wertzuweisung `x = 2` in der Funktion entsteht eine neue lokale Variable `x`. Sie hat zufällig denselben Bezeichner wie die globale Variable, ist aber eine andere Variable. In der Funktion wird die lokale Variable ausgegeben, außerhalb die globale. Die globale Variable bleibt unverändert.

5. (4 Punkte) In Java werden ganze Zahlen mit einer festen Anzahl von Bits im Zweierkomplement gespeichert.
   (a) Gib den Wertebereich für 8 Bit (`byte`) und für 16 Bit (`short`) an.
   (b) Gegeben ist der Java-Code

```
byte b = 127;
b++;
```

   Danach hat `b` den Wert -128. Erläutere, wie das zustande kommt.
   (c) In Python ergibt `127 + 1` immer `128`. Erläutere, warum.

Lösung:

(a) 8 Bit: $-2^7 = -128$ bis $2^7 - 1 = 127$. 16 Bit: $-2^{15} = -32\,768$ bis $2^{15} - 1 = 32\,767$.

(b) 127 ist der größte Wert, binär `01111111`. Plus 1 ergibt `10000000`. Im Zweierkomplement ist das höchste Bit das Vorzeichen, `10000000` bedeutet -128. Der Wertebereich wird überschritten, es gibt einen Überlauf ohne Fehlermeldung.

(c) In Python hat `int` keine feste Anzahl von Bits. Eine ganze Zahl kann beliebig groß werden, solange der Arbeitsspeicher reicht. Deshalb kann kein Überlauf passieren.

6. (4 Punkte) Gegeben ist der folgende Python-Code:

```
summe = 0
for i in range(10):
    summe = summe + 0.1
print(summe == 1.0)
```

   (a) Das Programm gibt `False` aus. Erläutere, warum.
   (b) Ändere die letzte Zeile so, dass geprüft wird, ob `summe` ungefähr gleich 1.0 ist.

Lösung:

(a) Gleitkommazahlen werden binär mit begrenzter Genauigkeit gespeichert. 0.1 lässt sich im Binärsystem nicht exakt darstellen. Bei jeder Addition entsteht ein kleiner Rundungsfehler, am Ende ist `summe` gleich `0.9999999999999999` und nicht genau 1.0.

(b)
```
print(abs(summe - 1.0) < 0.000001)
```

oder mit `import math`: `print(math.isclose(summe, 1.0))`

7. (5 Punkte) In einem Programm sollen folgende Daten gespeichert werden. Gib jeweils einen geeigneten Python-Datentyp (`int`, `float`, `bool` oder `str`) an und begründe kurz.
   (a) die Anzahl der Schüler eines Kurses
   (b) die Körpergröße in Metern
   (c) ob ein Schüler volljährig ist
   (d) ein Kontostand, mit dem exakt gerechnet werden soll
   (e) die Postleitzahl, z.B. 01067

   In Java soll die Einwohnerzahl der Erde (etwa 8 Milliarden) gespeichert werden. Zur Erinnerung: In Java speichert `int` eine ganze Zahl mit 32 Bit und `long` mit 64 Bit, beide im Zweierkomplement. Welcher der beiden Typen ist nötig? Begründe mit einer Abschätzung des Wertebereichs.

Lösung:

(a) `int`: eine Anzahl ist immer ganzzahlig.
(b) `float`: eine Größe hat Nachkommastellen.
(c) `bool`: es gibt nur die zwei Möglichkeiten ja und nein.
(d) `int`, und zwar in Cent: Mit `float` entstehen Rundungsfehler, mit ganzen Zahlen nicht.
(e) `str`: Mit einer Postleitzahl wird nicht gerechnet, und als `int` ginge die führende 0 verloren.

`long`: Der größte `int`-Wert ist $2^{31} - 1$. Mit $2^{10} \approx 10^3$ ist $2^{31} = 2 \cdot (2^{10})^3 \approx 2 \cdot 10^9$, also nur etwa 2 Milliarden. Das reicht nicht. Bei `long` ist $2^{63} = 8 \cdot (2^{10})^6 \approx 8 \cdot 10^{18}$, das reicht.

8. (4 Punkte) Gib für jede Anweisung an, was ausgegeben wird, und welchen Datentyp der ausgegebene Wert hat.

```
print(7 / 2)
print(7 // 2)
print(int(7.9))
print(int(-2.5))
print(float(4))
print(int('12') + 3)
print('12' + str(3))
print(True + True)
```

Lösung:

```
3.5     float   (/ liefert immer einen float)
3       int     (ganzzahlige Division)
7       int     (Nachkommastellen werden abgeschnitten)
-2      int     (abgeschnitten in Richtung 0)
4.0     float
15      int     (erst str -> int, dann Addition)
123     str     (Verkettung von zwei Strings)
2       int     (True wird als 1 gerechnet)
```

9. (5 Punkte) Das folgende Programm soll das Alter abfragen und ausgeben, wie alt man nächstes Jahr ist. Es enthält zwei Fehler, die mit Datentypen zu tun haben.

```
alter = input('Wie alt bist du? ')
naechstes = alter + 1
print('Nächstes Jahr bist du ' + naechstes + ' Jahre alt.')
```

   (a) Beschreibe die beiden Fehler und gib ein korrigiertes Programm an.

   (b) Was passiert im korrigierten Programm, wenn der Benutzer `16,5` oder `sechzehn` eingibt?

Lösung:

(a) `input()` liefert immer einen String. In Zeile 2 kann man zu einem String keine Zahl addieren (`TypeError`), der String muss zuerst mit `int()` umgewandelt werden. In Zeile 3 wird ein String mit einer Zahl verkettet (`TypeError`). Python wandelt die Zahl nicht automatisch in einen String um, man braucht `str()` oder einen f-String.

```
alter = int(input('Wie alt bist du? '))
naechstes = alter + 1
print('Nächstes Jahr bist du ' + str(naechstes) + ' Jahre alt.')
```

(b) In beiden Fällen kann `int()` den String nicht in eine ganze Zahl umwandeln. Das Programm bricht mit einem `ValueError` ab.

10. (5 Punkte) Gib für jede Anweisung an, ob eine implizite oder eine explizite Typumwandlung stattfindet, welchen Wert die Variable danach hat und welches Problem dabei auftritt (falls eines auftritt). Bei `e` genügt es, die Größenordnung des Wertes abzuschätzen.

   Zur Erinnerung: Ein `float` speichert höchstens etwa 16 Dezimalstellen.

```
a = 3 + 0.5
b = int(9.99)
c = int('3.5')
d = bool('False')
e = float(2 ** 60 + 1)
```

Lösung:

	Umwandlung	Wert	Problem
`a`	implizit	`3.5`	keines, `int` wird zu `float`
`b`	explizit	`9`	Informationsverlust: Nachkommastellen werden abgeschnitten, nicht gerundet
`c`	explizit	–	`ValueError`: `'3.5'` ist keine ganze Zahl
`d`	explizit	`True`	überraschend: jeder nicht leere String ergibt `True`
`e`	explizit	ca. 10^{18}	Genauigkeitsverlust: $2^{60} = (2^{10})^6 \approx (10^3)^6 = 10^{18}$ hat 19 Stellen, ein `float` speichert aber höchstens etwa 16. Die letzten Stellen gehen verloren, die +1 verschwindet. (Genauer Wert: `1.152921504606847e+18`)

11. (4 Punkte) In Java wird bei einem Type Cast in einen kleineren ganzzahligen Typ einfach nur der Teil der Bits übernommen, der in den Zieltyp passt. Ein `byte` hat 8 Bit und wird im Zweierkomplement gespeichert.
    (a) Bestimme den Wert von `(byte) 200`. Gib die Rechnung im Binärsystem an.
    (b) Bestimme den Wert von `(byte) 300`.
    (c) Beschreibe das Problem, das bei solchen Typumwandlungen auftritt.

Lösung:

(a) $200 = 11001000_2$. Das höchste Bit ist 1, also ist die Zahl im Zweierkomplement negativ: $-128 + 64 + 8 = -56$.

(b) $300 = 1\,00101100_2$ hat 9 Bit. Die unteren 8 Bit `00101100` werden übernommen: $32 + 8 + 4 = 44$.

(c) Liegt der Wert außerhalb des Wertebereichs des Zieltyps, entsteht ohne Fehlermeldung ein ganz anderer Wert, sogar mit anderem Vorzeichen.