1. (4 Punkte) Gegeben ist `t = (4, 7, 1, 9)`. Gib für jede Zeile die Ausgabe an oder ob ein Fehler auftritt. Die Zeilen werden unabhängig voneinander ausgeführt.

```
print(t[1:3])                    # Zeile 1
print(t + (2,))                  # Zeile 2
t[0] = 5                         # Zeile 3
t.append(3)                      # Zeile 4
a, b = t                         # Zeile 5
a, b, c, d = t; print(a + d)     # Zeile 6
print(type((5)), type((5,)))     # Zeile 7
```

Lösung:

```
Zeile 1: (7, 1)
Zeile 2: (4, 7, 1, 9, 2)     neues Tuple, t selbst bleibt gleich
Zeile 3: Fehler (TypeError), ein Tuple kann man nicht verändern
Zeile 4: Fehler (AttributeError), Tuples haben kein append
Zeile 5: Fehler (ValueError), 4 Werte, aber nur 2 Variablen
Zeile 6: 13
Zeile 7: <class 'int'> <class 'tuple'>   erst das Komma macht ein Tuple
```

2. (4 Punkte) Die Tageshöchsttemperaturen einiger Städte sind als Liste von Tuples gespeichert:

```
wetter = [('Stuttgart', 23), ('Hamburg', 17), ('Freiburg', 25), ('Berlin', 19)]
```

   (a) (2 Punkte) Gib mit einer `for`-Schleife alle Städte aus, in denen es wärmer als 20 Grad war. Entpacke die Tuples dabei im Schleifenkopf.

   (b) (2 Punkte) Bestimme die wärmste Stadt und ihre Temperatur, ohne `max` zu verwenden.

Lösung:

```
# a)
for stadt, temp in wetter:
    if temp > 20:
        print(stadt)                # Stuttgart, Freiburg

# b)
best_stadt, best_temp = wetter[0]
for stadt, temp in wetter:
    if temp > best_temp:
        best_stadt, best_temp = stadt, temp
print(best_stadt, best_temp)        # Freiburg 25
```

3. (4 Punkte) Gib an, wie das Dict `d` nach jeder Zeile aussieht. Gib in den Zeilen 5 bis 7 außerdem den Wert von `x`, `y` bzw. `z` an.

```
d = {'a': 1, 'b': 2}            # Zeile 1
d['c'] = 3                      # Zeile 2
d['a'] = d['a'] + d['c']        # Zeile 3
del d['b']                      # Zeile 4
x = d.get('b', 0)               # Zeile 5
y = 'c' in d                    # Zeile 6
z = list(d.values())            # Zeile 7
```

Lösung:

```
Zeile 1: {'a': 1, 'b': 2}
Zeile 2: {'a': 1, 'b': 2, 'c': 3}     neues Paar
Zeile 3: {'a': 4, 'b': 2, 'c': 3}     Wert von 'a' geändert
Zeile 4: {'a': 4, 'c': 3}
Zeile 5: unverändert, x = 0           'b' gibt es nicht mehr
Zeile 6: unverändert, y = True        in prüft die Schlüssel
Zeile 7: unverändert, z = [4, 3]
```

4. (5 Punkte) (a) (3 Punkte) Schreibe eine Funktion `wortanzahl(text)`, die ein Dict zurückgibt, in dem zu jedem Wort des Textes steht, wie oft es vorkommt. Groß- und Kleinschreibung soll keine Rolle spielen, die Wörter sind durch Leerzeichen getrennt.

   Beispiel: `wortanzahl('Der Hund und der Mann und die Katze')` liefert
   `{'der': 2, 'hund': 1, 'und': 2, 'mann': 1, 'die': 1, 'katze': 1}`

   (b) (2 Punkte) Gib mithilfe des Dicts alle Wörter aus, die mehr als einmal vorkommen, zusammen mit ihrer Anzahl.

Lösung:

```
# a)
def wortanzahl(text):
    anzahl = {}
    for w in text.lower().split():
        anzahl[w] = anzahl.get(w, 0) + 1
    return anzahl

# b)
anzahl = wortanzahl('Der Hund und der Mann und die Katze')
for w, n in anzahl.items():
    if n > 1:
        print(w, n)                  # der 2, und 2
```

5. (4 Punkte) Ein Teil des Morsealphabets ist in einem Dict gespeichert:

```
morse = {'a': '.-', 'e': '.', 'h': '....', 'l': '.-..', 'o': '---', 's': '...', 't': '-'}
```

Schreibe eine Funktion `morse_code(wort)`, die den Morsecode eines Wortes zurückgibt. Die Codes der einzelnen Buchstaben sollen durch ein Leerzeichen getrennt sein. Für einen Buchstaben, der nicht im Dict steht, soll ein `?` erscheinen. Großbuchstaben sollen wie Kleinbuchstaben behandelt werden.

Beispiele: `morse_code('Hallo')` liefert `'.... .- .-.. .-.. --'`, `morse_code('Post')` liefert `'? -- ... -'`.

Lösung:

```
def morse_code(wort):
    codes = []
    for c in wort.lower():
        codes.append(morse.get(c, '?'))
    return ' '.join(codes)
```

6. (6 Punkte) Die Bücher einer kleinen Bibliothek sind als Liste von Dicts gespeichert:

```
buecher = [
    {'titel': 'Momo', 'autor': 'Ende', 'jahr': 1973, 'preis': 12},
    {'titel': 'Tschick', 'autor': 'Herrndorf', 'jahr': 2010, 'preis': 11},
    {'titel': 'Die unendliche Geschichte', 'autor': 'Ende', 'jahr': 1979, 'preis': 15},
    {'titel': 'Der Vorleser', 'autor': 'Schlink', 'jahr': 1995, 'preis': 10},
]
```

Schreibe Python-Anweisungen, die

(a) (2 Punkte) die Titel aller Bücher ausgeben, die vor 2000 erschienen sind,

(b) (2 Punkte) den Gesamtpreis aller Bücher berechnen und ausgeben,

(c) (2 Punkte) ein Dict `pro_autor` erzeugen, das zu jedem Autor die Anzahl seiner Bücher enthält, hier also `{'Ende': 2, 'Herrndorf': 1, 'Schlink': 1}`.

Lösung:

```
# a)
for b in buecher:
    if b['jahr'] < 2000:
        print(b['titel'])

# b)
summe = 0
for b in buecher:
    summe = summe + b['preis']
print(summe)                          # 48

# c)
pro_autor = {}
for b in buecher:
    autor = b['autor']
    pro_autor[autor] = pro_autor.get(autor, 0) + 1
print(pro_autor)
```

7. (4 Punkte) Gegeben sind `a = {1, 2, 3, 4}` und `b = {3, 4, 5}`. Gib an, welchen Wert die folgenden Ausdrücke haben. (Bei Sets spielt die Reihenfolge der Elemente keine Rolle.)

```
a | b
a & b
a - b
b - a
{2, 1} <= a
set([3, 3, 1, 3])
len(set('banana'))
type({})
```

Lösung:

```
{1, 2, 3, 4, 5}      Vereinigung
{3, 4}               Schnittmenge
{1, 2}               in a, aber nicht in b
{5}                  in b, aber nicht in a
True                 {1, 2} ist Teilmenge von a
{1, 3}               Duplikate verschwinden
3                    die Buchstaben b, a, n
<class 'dict'>       {} ist ein leeres Dict, kein Set
```

8. (5 Punkte) In zwei Listen stehen die Anmeldungen für zwei AGs. Manche haben sich versehentlich doppelt angemeldet.

```
robotik = ['Anna', 'Ben', 'Carla', 'Ben', 'Emil']
theater = ['Dana', 'Carla', 'Anna', 'Fritz']
```

   (a) (3 Punkte) Berechne mit Sets und gib aus: wie viele verschiedene Personen insgesamt angemeldet sind, wer in beiden AGs ist und wer nur in der Robotik-AG ist.

   (b) (2 Punkte) Schreibe eine Funktion `hat_doppelte(liste)`, die `True` zurückgibt, wenn in der Liste mindestens ein Element mehrfach vorkommt, sonst `False`. Verwende ein Set und keine Schleife.

Lösung:

```
# a)
r = set(robotik)
t = set(theater)
print(len(r | t))        # 6
print(r & t)             # {'Anna', 'Carla'}
print(r - t)             # {'Ben', 'Emil'}

# b)
def hat_doppelte(liste):
    return len(set(liste)) < len(liste)
```

(Das Set enthält jedes Element nur einmal. Ist es kürzer als die Liste, gab es Duplikate.)

9. (5 Punkte) Welche Datenstruktur (Liste, Tuple, Dict oder Set) eignet sich jeweils am besten? Begründe kurz und gib ein kleines Beispiel in Python-Schreibweise an.
   (a) (1 Punkt) eine Farbe, gegeben durch ihre Rot-, Grün- und Blauwerte
   (b) (1 Punkt) ein Vokabelheft, in dem man zu einem deutschen Wort die englische Übersetzung nachschlägt
   (c) (1 Punkt) die Postleitzahlen aller Orte, aus denen schon Bestellungen kamen. Jede soll nur einmal gespeichert werden, und man will schnell prüfen können, ob eine Postleitzahl schon dabei ist.
   (d) (1 Punkt) die Einkäufe eines Kunden in der Reihenfolge, in der er sie in den Warenkorb legt
   (e) (1 Punkt) ein Schachbrett, auf dem man zu einer Position (Zeile, Spalte) die Figur findet, die dort steht

Lösung:

a) Tuple, drei zusammengehörende Werte, die Anzahl bleibt fest: `(255, 128, 0)`

b) Dict, zum Schlüssel (deutsches Wort) den Wert (Übersetzung) nachschlagen:
`{'Hund': 'dog', 'Katze': 'cat'}`

c) Set, keine Duplikate und `in` ist schnell: `{70173, 72072, 10115}`

d) Liste, die Reihenfolge zählt, die Liste wächst, Artikel dürfen mehrfach vorkommen:
`['Brot', 'Milch', 'Brot']`

e) Dict mit Tuples als Schlüssel: `{(0, 4): 'König', (1, 3): 'Bauer'}`