Informatik 1
Einführung in die Programmierung
WS 2023/24

Prof. Dr. Peter Thiemann
Institut für Informatik
Albert-Ludwigs-Universität Freiburg

- Für die Bearbeitung der Aufgaben haben Sie **150 Minuten** Zeit.
- Es sind **keine Hilfsmittel** wie Skripte, Bücher, Notizen oder Taschenrechner erlaubt. Des Weiteren sind alle elektronischen Geräte (wie z.B. Handys) auszuschalten. **Ausnahme: Fremdsprachige Wörterbücher** sind erlaubt.
- Falls Sie **mehrere Lösungsansätze** einer Aufgabe erarbeiten, markieren Sie deutlich, welcher gewertet werden soll. Die "Zielfunktion" darf nur einmal in der Abgabe definiert werden, alles andere muss auskommentiert oder gelöscht werden.
- Verwenden Sie **Typannotationen**, um die Typen der Parameter und des Rückgabewertes Ihrer Funktionen anzugeben. Verwenden Sie Typvariablen, falls die Funktion für beliebige Typen gelten soll. Fehlende oder falsche Typannotationen führen zu Punktabzug.
- Bearbeiten Sie die einzelnen Aufgaben in den vorgegebenen **Musterdateien**, z.B. `ex1_sequences.py`. **Falsch benannte Funktionen** werden nicht bewertet. **Neu erstellte Dateien** werden nicht bewertet.
- **Die Zielfunktionen dürfen ihre Eingaben nicht verändern**, d.h. Methoden wie `list.remove` dürfen nicht auf die Eingaben angewendet werden; **es sei denn**, die Aufgabenstellung fordert explizit die Eingabe zu verändern.
- Intern darf Ihre Implementierung den vollen Sprachumfang verwenden; es sei denn, die Aufgabenstellung schließt etwas aus.
- **Sie dürfen keine Module importieren.** Alle Imports, die benutzt werden dürfen/müssen sind bereits vorgegeben. Zum Lösen der Aufgaben sind keine weiteren Importe/Module notwendig.

	Erreichbare Punkte	**Erzielte Punkte**	**Nicht bearbeitet**
Aufgabe 1	10		
Aufgabe 2	15		
Aufgabe 3	20		
Aufgabe 4	10		
Aufgabe 5	20		
Aufgabe 6	15		
Aufgabe 7	15		
Aufgabe 8	15		
Gesamt	120		

Aufgabe 1 (Sequence; Punkte: 10)**.**

Betrachten Sie folgende Funktion $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$f(x, y) = \begin{cases} \frac{x}{2} & x \text{ ist gerade} \\ x + 2 \cdot y - 1 & \text{sonst} \end{cases}$$

Sei a eine positive natürliche Zahl, dann definieren wir die Folge $(s_i^a)_{i \in \mathbb{N}}$ durch

$$\begin{aligned} s_0^a &= a \\ s_1^a &= f(s_0^a, a) \\ s_2^a &= f(s_1^a, a) \\ s_3^a &= f(s_2^a, a) \\ &\dots \\ s_n^a &= f(s_{n-1}^a, a) \\ &\dots \end{aligned}$$

Schreiben Sie eine Funktion `count_iterations`, welche zwei positive natürliche Zahlen a und b als Argumente nimmt und das kleinste $n \geq 0$ zurückgibt, für das $s_n^a < b$ gilt.
Beispiel:

```
>>> count_iterations(7, 6)
3
```

Da im Beispiel $a = 7$ und $b = 6$ ist, suchen wir die erste Zahl in der Folge $(s_i^7)_{i \in \mathbb{N}}$ die kleiner 6 ist:

$$\begin{aligned} s_0^7 &= 7 && \geq 6 \\ s_1^7 &= f(7, 7) = 20 && \geq 6 \\ s_2^7 &= f(20, 7) = 10 && \geq 6 \\ s_3^7 &= f(10, 7) = 5 && < 6 \end{aligned}$$

Dies ist der Fall für s_3^7, also wird 3 zurückgegeben.

Aufgabe 2 (Dictionaries und Sets; Punkte: 15)**.**

Ein Straßenbahnnetz (`dict`) ordnet Straßenbahnlinien (`int`) die Mengen ihrer Stationen (`str`) zu. Zum Beispiel:

```
>>> network = {
...     1: {"Hauptbahnhof", "Paduaallee", ...},
...     2: {"Hauptbahnhof", "Stadttheater", ...},
...     3: {"Am Lindenwäldle", "Hauptbahnhof", ...},
...     5: {"Europaplatz", "Stadttheater", ...},
... }
```

(a) (5 Punkte) Schreiben Sie eine Funktion `lines`, die ein Straßenbahnnetz `network` und eine Station `station` als Argumente nimmt und die Menge aller Straßenbahnlinien zurückgibt, die an `station` halten.

(b) (5 Punkte) Schreiben Sie eine Funktion `invert`, die ein Straßenbahnnetz `network` als Argument nimmt und ein neues Dictionary zurückgibt. Dieses soll jeder Station in `network` die Menge an Straßenbahnlinien zuordnen, die an dieser Station halten.

(c) (5 Punkte) Schreiben Sie eine Funktion `add_line`, die ein Straßenbahnnetz `network` und eine Menge von Stationen `stations` als Argumente nimmt. Die Funktion soll eine neue Straßenbahnlinie, mit den Stationen `stations` in das Straßenbahnnetz einfügen. Die Nummer der Straßenbahnlinie soll dabei gerade die kleinste positive Zahl (`> 0`) sein, welche noch keine Nummer einer anderen Straßenbahnlinie in `network` ist. Die Funktion soll nichts zurückgeben, sondern stattdessen `network` verändern.

Beispiele:

```
>>> network = {
...     1: {"Hauptbahnhof", "Paduaallee", "Stadttheater"},
...     2: {"Hauptbahnhof", "Stadttheater", "Hornusstraße", "Johanneskirche"},
...     3: {"Am Lindenwäldle", "Hauptbahnhof", "Stadttheater", "Johanneskirche"},
...     5: {"Europaplatz", "Stadttheater", "Am Lindenwäldle"},
... }
>>> lines(network, "Hauptbahnhof")
{1, 2, 3}
>>> invert(network)
{'Paduaallee': {1}, 'Stadttheater': {1, 2, 3, 5}, 'Hauptbahnhof': {1, 2, 3},
↪  'Johanneskirche': {2, 3}, 'Hornusstraße': {2}, 'Am Lindenwäldle': {3, 5}, 'Europaplatz':
↪  {5}}
>>> add_line(network, {
...         "Technische Fakultät","Hauptbahnhof",
...         "Stadttheater","Europaplatz","Hornusstraße",
...     }
... )
>>> network[4]
{'Hornusstraße', 'Hauptbahnhof', 'Technische Fakultät', 'Stadttheater', 'Europaplatz'}
```

Aufgabe 3 (Strings; Punkte: 20)**.**

(a) (10 Punkte) Schreiben Sie eine Funktion `s1_in_s2`, die zwei Strings `s1` und `s2` als Argumente nimmt und genau dann `True` zurückgibt, wenn alle Zeichen aus `s1` in der selben Reihenfolge in `s2` vorkommen, ansonsten `False`.

Beispiele:

```
>>> s1_in_s2("fn", "function")
True
>>> s1_in_s2("ufnction", "function")
False
>>> s1_in_s2("fnn", "function")
True
>>> s1_in_s2("fcc", "function")
False
```

(b) (10 Punkte) Schreiben Sie eine Funktion `split_text`, die einen String als Argument nimmt, diesen in Worte und Nicht-Worte aufteilt und als Liste zurückgibt:

- Ein Wort ist ein String, der ausschließlich Buchstaben enthält (a-z und A-Z).
- Ein Nicht-Wort ist ein String, der ausschließlich Zeichen enthält, die keine Buchstaben sind.

Beispiele:

```
>>> split_text("You're a lizard, Harry!")
['You', "'", 're', ' ', 'a', ' ', 'lizard', ', ', 'Harry', '!']
>>> split_text("Luke! I'm your father!!")
['Luke', '! ', 'I', "'", 'm', ' ', 'your', ' ', 'father', '!!']
>>> split_text("*Stay away from her, you $!#@!*")
['*', 'Stay', ' ', 'away', ' ', 'from', ' ', 'her', ', ', 'you', '
↪  $!#@!*']
```

Die einzelnen Worte und Nicht-Worte müssen dabei so lang wie möglich sein. Zum Beispiel, soll `split_text("hello  world")` die Liste `["hello", "  ", "world"]` zurückgeben und nicht `["he", "l", "lo", " ", " ", "wor", "ld"]`.

Hinweis: Verwenden Sie die `str`-Methode `isalpha` um zu prüfen, ob ein Zeichen ein Buchstabe ist.

Aufgabe 4 (Zustandsautomat; Punkte: 10)**.**

In der folgenden Aufgabe sollen Sie eine Instanz der Datenklasse `Automaton` erstellen.

```
@dataclass
class Automaton[Q]:
    E: frozenset[str]              # Eingabealphabet
    delta: Callable[[Q, str], Q]   # Übergangsfunktion
    q0: Q                          # Startzustand
    F: frozenset[Q]                # Akzeptierende Zustände

    def accept(self, input: str) -> bool:
        state = self.q0
        for c in input:
            state = self.delta(state, c)
        return state in self.F
```

Das Eingabealphabet des Zustandsautomaten soll aus den Zeichen {`'a'`, `'b'`} bestehen. Die Zustände des Automaten, der Start- und Endzustand, sowie die Übergänge zwischen den Zuständen sind durch Abbildung 1 definiert.

(a) (3 Punkte) Schreiben Sie ein Enum `State` für die Zustände `q0`, `q1` und `q2`.

(b) (6 Punkte) Schreiben Sie eine Funktion `delta`, die einen State `state` und einen Eingabe-String `input` als Argumente nimmt und den jeweiligen Folgezustand von `state` basierend auf Abbildung 1 zurückgibt. Verwenden Sie hierzu Pattern-Matching.

(c) (1 Punkt) Schreiben Sie anschließend die Funktion `automaton`, die eine Instanz des beschriebenen Automaten zurückgibt.

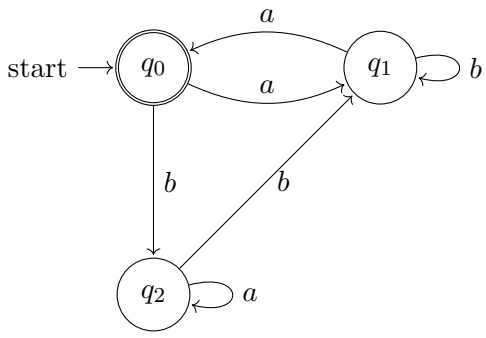

Abbildung 1: Zustandsdiagramm des Automaten

Aufgabe 5 (Dataclasses; Punkte: 20)**.**

In der folgenden Aufgabe sollen Sie Datenklassen nutzen. Achten Sie darauf, so wenig wie möglich doppelt zu schreiben, indem Sie Vererbung, Methodenüberschreibung und `super` benutzen.

(a) (8 Punkte) Erstellen Sie eine Datenklasse `Vehicle` mit den ganzzahligen Attributen `seats` (Anzahl der Sitze), `hp`, `ccm` und `weight`.
Bei der Instanziierung soll sichergestellt werden, dass die Anzahl der Sitze größer als Null und kleiner als Zehn und die restlichen Attribute jeweils größer als Null sind.

Fügen Sie die Methode `fun_factor` hinzu, die zurückgibt wieviel Spaß das Fahren eines `Vehicle` macht. Die Berechnung dafür ist `(10 * hp + ccm) / weight`.

Implementieren Sie den Operator `>`, der zwei `Vechicle` vergleicht.
Der Vergleich `vehicle1 > vehicle2` soll genau dann `True` ergeben, wenn `vehicle1` einen höheren `fun_factor` hat als `vehicle2`, ansonsten `False`.

(b) (6 Punkte) Schreiben Sie eine Datenklasse `Car`, die von `Vehicle` erbt. Ein `Car` soll zusätzlich das boolesche Attribut `spoiler` besitzen, das angibt, ob ein `Car` einen Spoiler hat oder nicht. Wenn ein `Car` einen Spoiler hat, so steigt der `fun_factor` um `0.2`.

(c) (6 Punkte) Schreiben Sie eine Datenklasse `Motorcycle`, die ebenfalls von `Vehicle` erbt. Ein `Motorcycle` verfügt über ein zusätzliches Attribut `sidecar`, das angibt, ob das `Motorcycle` einen Beiwagen hat oder nicht. Bei der Instanziierung soll sichergestellt werden, dass die Anzahl der Sitze Eins oder Zwei ist. Hat das Motorcycle einen Beiwagen, so kann die Anzahl der Sitze Zwei oder Drei sein. Ein `Motorcycle` ohne Beiwagen hat einen drei mal höheren `fun_factor` als ein herkömmliches `Vehicle`. Hat ein `Motorcycle` einen Beiwagen, so ist der `fun_factor` nur um den Faktor `2.4` höher.

Aufgabe 6 (Rekursion; Punkte: 15)**.**

Im Folgenden betrachten wir binäre Bäume, die wie in der Vorlesung über eine generische Datenklasse `Node` implementiert sind:

```
@dataclass
class Node[T]:
    mark: T
    left: 'Tree[T]' = None
    right: 'Tree[T]' = None

type Tree[T] = Optional[Node[T]]  # trees can be empty
```

(a) (8 Punkte) Schreiben Sie eine Funktion `filter_tree`, die eine Funktion `f` und einen beliebigen Baum `tree` als Argumente nimmt und die Liste der Markierung im Baum `tree` zurückgibt, für die `f` `True` zurückgibt. Die Liste soll dabei in *Pre-Order*-Reihenfolge erstellt werden.

Beispiel:

```
>>> example1 = Node("zero",
...                  Node("one"),
...                  Node("two",
...                       Node("three"),
...                       Node("four")
...                       )
...                  )
>>> filter_tree(lambda x: 'e' in x, example1)
['zero', 'one', 'three']
>>> filter_tree(lambda x: 'o' in x, example1)
['zero', 'one', 'two', 'four']
>>> filter_tree(lambda _: False, example1)
[]
>>> filter_tree(lambda _: True, None)
[]
```

(b) (7 Punkte) Schreiben Sie eine Funktion `mirror_tree`, die einen beliebigen Baum `tree` als Argument nimmt. Die Funktion soll rekursiv für jeden Knoten des Baums das linke mit dem rechten Kind tauschen. Dabei soll kein neuer Baum zurückgegeben werden, sondern `tree` selbst verändert werden.

Beispiel:

```
>>> example2 = Node(0, Node(5, Node(3)), Node(1))
>>> mirror_tree(example2)
>>> example2
Node(mark=0, left=Node(mark=1, left=None, right=None),
↪  right=Node(mark=5, left=None, right=Node(mark=3, left=None,
↪  right=None)))
```

Aufgabe 7 (Generatoren; Punkte: 15)**.**

Vermeiden Sie unnötigen Speicherverbrauch: Funktionen, die iterierbare Objekte (Iterables) als Argument nehmen, dürfen diese nicht unnötigerweise in eine Liste umwandeln.

(a) (5 Punkte) Schreiben Sie eine Generator-Funktion `drop`, die ein beliebiges iterierbares Objekt `xs` und eine ganze Zahl `n` als Argumente nimmt. Der Generator soll die ersten `n` Elemente von `xs` überspringen, und nur die übrigen Elemente von `xs` generieren. Hat `xs` weniger als `n` Elemente, soll der Generator keinen Wert generieren.

Beispiele:

```
>>> list(drop([2, 4, 6, 8, 10, 12], 3))
[8, 10, 12]
>>> list(drop([True, False, False], 0))
[True, False, False]
>>> list(drop("abcde", 8))
[]
```

(b) (5 Punkte) Schreiben Sie eine Generator-Funktion `split`, die ein beliebiges iterierbares Objekt `xs` und einen Separator `sep` als Argumente nimmt. Der Separator hat dabei den gleichen Typ, wie die Elemente von `xs`. Die Generator-Funktion soll Listen derjenigen Teile von `xs` generieren, die durch `sep` getrennt sind.

Beispiele:

```
>>> list(split([1, 5, 3, 4, 9, 3, 5], 3))
[[1, 5], [4, 9], [5]]
>>> list(split("mississippi", "i"))
[['m'], ['s', 's'], ['s', 's'], ['p', 'p'], []]
```

(c) (5 Punkte) Schreiben Sie eine Generator-Funktion `apply_pairs`, die ein beliebiges iterierbares Objekt `xs` und eine zweistellige Funktion `f` als Argumente nimmt. Die Generator-Funktion soll `f` der Reihe nach auf **alle** Paare aufeinanderfolgender Werte in `xs` anwenden und das jeweilige Ergebnis generieren.

Beispiele:

```
>>> sub = lambda x, y: x - y
>>> eq = lambda x, y: x == y
>>> list(apply_pairs([5, 2, 7, 9, 1], sub))
[3, -5, -2, 8]
>>> list(apply_pairs("abaabbc", eq))
[False, False, True, False, True, False]
>>> list(apply_pairs([1], sub))
[]
```

Aufgabe 8 (Funktionale Programmierung und Comprehensions; Punkte: 15)**.**

Implementieren Sie die Funktionen aus folgenden Teilaufgaben im funktionalen Stil - mit einem Rumpf, der aus genau einer `return`-Anweisung besteht.

(a) (5 Punkte) Schreiben Sie eine Funktion `sum_0`, die eine Liste von einstelligen Funktionen `fs` (von `float` nach `float`) als Argument nimmt und die Summe der Funktionen angewendet auf den Wert `0.0` zurückgibt.

Beispiel:

```
>>> f = lambda x: x + 1
>>> g = lambda x: x**2
>>> sum_0([f, g])
1
```

(b) (5 Punkte) Schreiben Sie eine Funktion `extensionally_equal`, die zwei einstellige Funktionen `f` und `g` und eine beliebige Liste `dom` als Argumente nimmt. Die Funktion soll genau dann `True` zurückgeben, wenn für alle Werte `x` in `dom` gilt, dass `f(x)` gleich `g(x)` ist, ansonsten `False`.

Beispiel:

```
>>> f = lambda x: x * 2 - 1
>>> g = lambda y: y * (4 / 2) - 1
>>> extensionally_equal(f, g, [1, 2, 3, 4, 5])
True
```

(c) (5 Punkte) Eine Matrix kann durch eine Liste von Listen von Gleitkommazahlen dargestellt werden. Schreiben Sie eine Funktion `map_matrix`, die eine einstellige Funktion `f` (von `float` nach `float`) und eine Matrix `m` als Argumente nimmt und eine Matrix zurückgibt, in der die Funktion `f` auf jede Zahl der Matrix `m` angewandt wurde. Verwenden Sie bei der Implementierung **keine** List-Comprehensions, dafür aber mehrere Aufrufe der `map`- und `list`-Funktionen.

Beispiel:

```
>>> example = [[1, 2, 3], [4, 5, 6]]
>>> map_matrix(lambda x: x * 2, example)
[[2, 4, 6], [8, 10, 12]]
```