Klausur 06. August 2024

Einführung in die Programmierung
SS 2024

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_sequence.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	15		
Aufgabe 2	20		
Aufgabe 3	20		
Aufgabe 4	20		
Aufgabe 5	15		
Aufgabe 6	20		
Aufgabe 7	20		
Aufgabe 8	20		
Gesamt	150		

Aufgabe 1 (Sequence; Punkte: 15)**.**

Wir definieren die Folge `juggle` für ein bereits gegebenes $a_0 \in \mathbb{N}$ wie folgt:

$$a_{k+1} = \begin{cases} \lfloor \sqrt{a_k} \rfloor & \text{wenn } a_k \text{ gerade} \\ \left\lfloor a_k^{3/2} \right\rfloor & \text{wenn } a_k \text{ ungerade} \end{cases}$$

Schreiben Sie eine Funktion `juggle`, die einen Basiswert a_0 und eine Schrittzahl $k \in \mathbb{N}$ als Argumente nimmt und den Wert a_k der Folge berechnet.
Beispiel:

```
>>> juggle(42, 0)
42
>>> juggle(16, 1)
4
>>> juggle(3, 1)
5
>>> juggle(3, 6)
1
```

Hinweis: Benutzen Sie die Funktionen `floor` (zu deutsch: abrunden) und `sqrt` (zu deutsch: Quadratwurzel) aus dem `math` Modul.
Hinweis: Sie können den Typ `int` für natürliche Zahlen verwenden, und davon ausgehen, dass ausschließlich positive Zahlen in die Funktion gegeben werden.

Aufgabe 2 (Dictionaries; Punkte: 20)**.**

Wikipedia-Artikel verlinken zu anderen Wikipedia-Artikeln. Dieses Netzwerk an Verlinkungen kann als Dictionary (`dict`) dargestellt werden, in dem jedem Artikel (`str`) eine Menge (`set`) von Artikeln zugeordnet wird, auf die dieser *direkt* verlinkt.
Beispiel:

```
>>> network = {
...   'Python': {'Programming', 'Computer Science'},
...   'Programming': {'Computer Science', 'Software Engineering'},
...   'Computer Science': {'Software Engineering'},
...   'Software Engineering': set()
... }
```

(a) (7 Punkte) Schreiben Sie eine Funktion `reachable_from`, die ein solches Netzwerk an Verlinkungen `network` nimmt und ein Dictionary zurückgibt, das für jeden Artikel die Menge an Artikeln enthält, die auf diesen Artikel *direkt* verlinken. Wird ein Artikel von keinem anderen Artikel verlinkt, soll er dennoch in der Rückgabe enthalten sein.

Beispiel:

```
>>> reachable_from({})
{}
>>> reachable_from(network)
{'Python': set(), 'Programming': {'Python'}, 'Computer Science':
↪  {'Programming', 'Python'}, 'Software Engineering': {'Programming',
↪  'Computer Science'}}
```

(b) (7 punkte) Schreiben Sie eine Funktion `number_of_outgoing_links`, die eine Menge von Artikeln `articles` und ein Netzwerk an Verlinkungen `network` nimmt und die Anzahl aller Artikel berechnet, die *direkt* (in einem Schritt) von einem beliebigen Artikel aus `articles` erreicht werden können.

Beispiel:

```
>>> number_of_outgoing_links(set(), {})
0
>>> number_of_outgoing_links({"Ungültiger Artikel", "Computer
↪  Science"}, network)
1
>>> number_of_outgoing_links({"Programming", "Computer Science"},
↪  network)
2
```

(c) (6 Punkte) Schreiben Sie eine Funktion `remove_article`, die einen Artikel `article` und ein Netzwerk an Verlinkungen `network` nimmt. Die Funktion soll ein *neues* Netzwerk an Verlinkungen zurückgeben, in dem alle Vorkommnisse von `article` entfernt wurden. Alle anderen Artikel und Verlinkungen sollen in das neue Dictionary übernommen werden.

Beispiel:

```
>>> remove_article("Ungültiger Artikel", network)
{'Python': {'Programming', 'Computer Science'}, 'Programming':
↪  {'Computer Science', 'Software Engineering'}, 'Computer Science':
↪  {'Software Engineering'}, 'Software Engineering': set()}
```

```
>>> remove_article("Python", network)
{'Programming': {'Computer Science', 'Software Engineering'}, 'Computer
↪  Science': {'Software Engineering'}, 'Software Engineering': set()}
>>> remove_article("Computer Science", network)
{'Python': {'Programming'}, 'Programming': {'Software Engineering'},
↪  'Software Engineering': set()}
```

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

Hinweis: Die folgenden Funktionen auf Strings können nützlich zum Lösen der Aufgabe sein:

```
>>> "Hallo Welt".replace("l", "x")  # replace
'Haxxo Wext'
>>> "??".join(["Hallo", "Welt"])  # join
'Hallo??Welt'
>>> "Hallo$Welt".split("$")  # split
['Hallo', 'Welt']
>>> "\n Hallo Welt \n".strip()  # strip
'Hallo Welt'
>>> int("-189")  # int
-189
```

Hinweis: In dieser Aufgabe dürfen Sie die Python-Funktion `eval` nicht benutzen.

(a) (10 Punkte) Schreiben Sie eine Funktion `parse_matrix`, die einen String `sm` in eine Liste von Listen von ganzen Zahlen (Matrix) umwandelt. Sie dürfen davon ausgehen, dass `sm` für $n, m \in \mathbb{N}$ immer die Struktur $[[i_{11}, i_{12}, .., i_{1m}], .., [i_{n1}, .., i_{nm}]]$ hat, wobei an *beliebigen* Stellen Leerzeichen vorkommen dürfen und alle i_{nm} gültige `int` sind.

Beispiele:

```
>>> parse_matrix("[ []]")
[[]]
>>> parse_matrix("[[1]]")
[[1]]
>>> parse_matrix("[[-129], [4]]")
[[-129], [4]]
>>> parse_matrix("[  [1, 20, 3 0], [4,5,  6]] ")
[[1, 20, 30], [4, 5, 6]]
```

(b) (10 Punkte) Schreiben Sie eine Funktion `pretty`, die eine Liste von Listen von ganzen Zahlen (Matrix) `m` nimmt und diese in einen formatieren String umwandelt und zurückgibt. Die Funktion soll jede Zeile der Matrix in einer neuen Zeile des Strings darstellen, wobei die Werte in jeder Zeile durch *genau* ein Leerzeichen getrennt sind. Ansonsten dürfen keine Leerzeichen vorkommen.

Hinweis: Die neue Zeile wird im Beispiel als `\n` dargestellt, im Terminal allerdings als echte neue Zeile.

Beispiel:

```
>>> pretty([[]])
''
>>> pretty([[1]])
'1'
>>> pretty([[1], [-3]])
'1\n-3'
>>> pretty([[1, 2], [3, 4]])
'1 2\n3 4'
```

Aufgabe 4 (Dataclasses; Punkte: 20)**.**

(a) (12 Punkte)

Schreiben Sie eine Datenklasse `Pokemon` mit dem ganzzahligen Attribut `attack_damage` (Angriffsschaden) und dem privaten ganzzahligen Attribut `__hp` (Lebenspunkte), das mit `100` initialisiert wird. Stellen Sie mithilfe eines `assert` sicher, dass nur Pokemon mit Angriffsschaden größer als `0` erstellt werden können.

Definieren Sie Getter und Setter für das private Attribut `__hp`. Der Setter soll für negative Lebenspunkte die Lebenspunkte auf 0 setzen.

Definieren Sie die Methode `fight`, die ein anderes Pokemon `other` als Argument nimmt. Bei einem Kampf wird beiden Pokemon von ihren Lebenspunkten der Angriffsschaden des jeweils anderen abgezogen. Ein Kampf dauert solange an, bis mindestens eins der beiden Pokemon keine Lebenspunkte mehr hat.

(b) (8 Punkte)

Hinweis: Vermeiden Sie in dieser Aufgabe unnötiges Duplizieren von Programmcode.

Schreiben Sie eine Datenklasse `LevelPokemon`, die von `Pokemon` erbt.

Erweitern Sie die Datenklasse `LevelPokemon` um das Attribut `level` (int).

Stellen Sie mithilfe eines `assert` sicher, dass nur Pokemon mit einem Level von mindestens 0 erstellt werden können.

Die Methode `fight` soll zuerst die Pokemon gegeneinander kämpfen lassen, dann aber Prüfen ob der Kampf unentschieden ausgegangen ist. Hat ein `LevelPokemon` gewonnen, soll sich das Level des Gewinners um `1` erhöhen.
Verwenden Sie hierzu Pattern Matching.

Definieren Sie den Operator `<` (`__lt__`) auf `LevelPokemon`. Dabei sollen zunächst die Level, dann die Lebenspunkte verglichen werden.

Aufgabe 5 (Zustandsautomat; Punkte: 15)**.**

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` und `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`, `q2` und `q3`.

(b) (6 Punkte) Schreiben Sie eine Funktion `delta`, die einen State `state` und einen einstelligen, gültigen (aus dem Eingabealphabet) Eingabe-String `input` als Argumente nimmt und den jeweiligen Folgezustand von `state` basierend auf Abbildung 1 zurückgibt.

Verwenden Sie hierzu Pattern Matching.

(c) (3 Punkt) Schreiben Sie anschließend die Funktion `automaton`, die eine Instanz des beschriebenen Automaten zurückgibt.

Beispiel:

```
>>> automaton().accept("aba")
True
>>> automaton().accept("ab")
False
```

(d) (3 Punkt) Schreiben Sie eine Funktion `complement`, die eine Instanz eines Automaten zurückgibt, der genau die Eingaben akzeptiert, die der Automat aus der Abbildung *nicht* akzeptiert.

Beispiel:

```
>>> complement().accept("aba")
False
>>> complement().accept("ab")
True
```

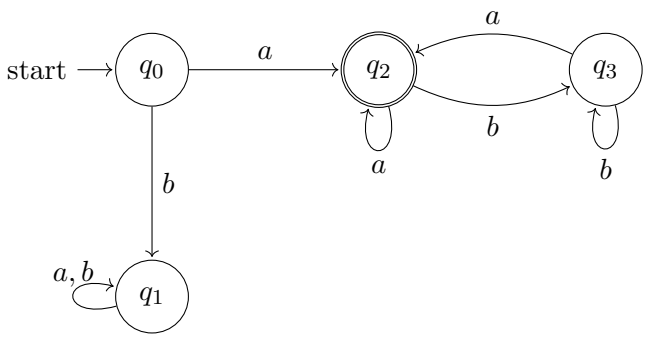

Abbildung 1: Zustandsdiagramm des Automaten

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

Binary Decision Diagrams (BDDs) sind *nicht-leere* Binärbäume zur Repräsentation von Funktionen, die eine Liste von Wahrheitswerten $xs = [x_0, x_1, \ldots, x_{n-1}]$ auf einen Wert des Typs `T` abbilden. Die inneren Knoten (`Node`) des BDDs repräsentieren die Entscheidungen, die anhand der Werte der Eingabevariablen getroffen werden. Die Blattknoten (`Leaf`) enthalten die möglichen Funktionswerte.
Folgende Definitionen sind gegeben und sollen zum Lösen dieser Aufgabe verwendet werden:

```python
@dataclass
class Node[T]:
    index: int  # 0 <= index < n
    left: 'BDD[T]'
    right: 'BDD[T]'

@dataclass
class Leaf[T]:
    value: T

type BDD[T] = Node[T] | Leaf[T]
```

(a) (10 Punkte) Schreiben Sie eine Funktion `height`, die einen BDD `bdd` nimmt und die Höhe des BDDs berechnet. Die Höhe eines Baumes ist definiert als die Anzahl von *Kanten* auf dem längsten Pfad von der Wurzel zu einem *beliebigen* Blatt.

Verwenden Sie Pattern matching und rekursive Funktionsaufrufe.

Hinweis: Sie können die Funktion `max` verwenden.

```
>>> height(Leaf("Hallo Welt"))
0
>>> height(Node(3, Leaf(True), Leaf(False)))
1
>>> height(Node(2, Leaf("A"), Node(5, Node(1, Leaf("B"), Leaf("C")),
↪  Leaf("D"))))
3
```

(b) (10 Punkte) Schreiben Sie eine Funktion `eval`, die einen BDD `bdd` und eine Liste von Wahrheitswerten (`bool`) `xs` als Argumente nimmt. Die Funktion soll den Wert des `bdd` für die Eingabe `xs` berechnen. Die Auswertung der inneren Knoten erfolgt, indem der Wert an der Listen-Position `index` des Knotens in `xs` betrachtet wird. Wenn der Wert `True` ist, wird der linke Teilbaum weiter ausgewertet, ansonsten der rechte Teilbaum. Bei Erreichen eines Blatts wird der dort vorgefundene `value` als Funktionswert zurückgegeben.

Verwenden Sie Pattern Matching und rekursive Funktionsaufrufe.

Hinweis: Sie dürfen annehmen, dass `index` für jeden Blattknoten stets kleiner als die Länge der Eingabeliste `xs` ist.

Beispiel:

```
>>> eval(Leaf("Hallo Welt"), [])
'Hallo Welt'
>>> eval(Node(2, Leaf(True), Leaf(False)), [True, False, True])
```

```
True
>>> eval(Node(2, Leaf("A"), Node(5, Node(1, Leaf("B"), Leaf("C")),
↪  Leaf("D"))), [False, True, False, False, False, True])
'B'
```

Aufgabe 7 (Generatoren; Punkte: 20)**.**

Hinweis: Vermeiden Sie unnötigen Speicherverbrauch: Funktionen, die Iteratoren als Argument nehmen, dürfen diese nicht unnötigerweise in eine Liste umwandeln.

(a) (10 Punkte) Schreiben Sie eine Generator-Funktion `windowed`, die einen Iterator `it` und eine ganze Zahl `size` als Argumente nimmt und Fenster (Listen) der Größe `size` generiert. Das erste Fenster enthält die ersten `size` Elemente des Iterators `it`. Danach bewegt sich das Fenster in jedem Schritt um eine Position nach rechts. Enthält der Iterator weniger als `size` Elemente soll kein Window generiert werden.

Beispiel:

```
>>> it = windowed(iter([1, 2, 3, 4]), 3)
>>> next(it)
[1, 2, 3]
>>> next(it)
[2, 3, 4]
```

(b) (10 Punkte) Schreiben Sie eine Generator-Funktion `moving_binary_sum`, die einen Iterator `it` als Argument nimmt und die Summen aller benachbarten Argumente generiert.

Beispiel:

```
>>> it = moving_binary_sum(iter([1, 2, 3, 4]))
>>> next(it)  # 1 + 2
3
>>> next(it)  # 2 + 3
5
>>> next(it)  # 3 + 4
7
```

Aufgabe 8 (Funktionale Programmierung und Comprehensions; Punkte: 20)**.**

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

(a) (7 Punkte) Schreiben Sie eine Funktion `unwrap_or_else`, die einen optionalen Wert `x` vom Typ `A` und eine Funktion `orelse` als Argument nimmt. Die Funktion `orelse` hat keine Argumente und gibt auch etwas vom Typ `A` zurück.

Die Funktion `unwrap_or_else` soll `x` zurück geben, wenn `x` nicht `None` ist und den Rückgabewert von `orlese` sonst. Beispiel:

```
>>> unwrap_or_else(42, lambda: 0)
42
>>> unwrap_or_else(None, lambda: 42)
42
```

(b) (7 Punkte) Schreiben Sie eine Funktion `strange_map` die eine Liste von Funktionen `fs` und einen Wert `x` vom Typ `A` als Argumente nimmt. Die Funktionen in der Liste `fs` nehmen ein Argument vom Typ `A` und haben Rückgabetyp `B`.

Die Funktion `strange_map` soll jede Funktion der Eingabeliste auf `x` anwenden und die Liste der Ergebnisse zurückgeben. Beispiel:

```
>>> strange_map([lambda x: str(x), lambda x: str(x + 1)], 1)
['1', '2']
>>> strange_map([lambda x: x * 2, lambda x: x // 2], 4)
[8, 2]
```

(c) (6 Punkte) Schreiben Sie eine Funktion `curry`, die eine Funktion `f` als Argument nimmt, die zwei Argumente mit Typen `A` und `B` nimmt und Rückgabetyp `C` hat.

Die Funktion `curry` soll eine neue Funktion zurückgeben, die ein Argument vom Typ `A` nimmt und eine Funktion zurückgibt, die ein Argument vom Typ `B` nimmt und Rückgabetyp `C` hat. Die zurückgegebene Funktion soll sich genau so verhalten, wie `f`.

Beispiel:

```
>>> curry(lambda a, b: a + b)(1)(2)
3
>>> curry(lambda a, b: a[b])([1, 2])(0)
1
```