Einführung in die Programmierung
WS 2024/25

Prof. Dr. Peter Thiemann
Institut für Informatik
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 geforderten Funktionen dürfen 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	15		
Aufgabe 5	20		
Aufgabe 6	20		
Aufgabe 7	20		
Aufgabe 8	20		
Gesamt	150		

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

Die Quadratwurzel einer beliebigen Zahl $s \in \mathbb{R}$ mit $s \geq 0$ kann berechnet werden, indem der Schnittpunkt der Funktion

$$f(x) = x^2 - s$$

mit der x-Achse bestimmt wird.
Numerisch kann dies mit Hilfe des Newton-Verfahrens angenähert werden, indem man mit einem Anfangswert $x_0 = s/2$ startet und nach folgender Vorschrift iteriert:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Dabei ist $f'(x) = 2x$ die Ableitung der Funktion f. Wir iterieren dabei solange weiter, bis der Funktionswert $f(x_n)$ nahe genug an 0 liegt, d.h. wenn $|f(x_n)| < \epsilon$, wobei ϵ eine vorgegebene Fehlertoleranz ist.

Implementieren Sie eine Funktion `sqrt_newton`, die zwei Gleitkommazahlen `s` und `eps` als Argumente erhält. Die Funktion soll die Quadratwurzel von `s` mit Hilfe des oben beschriebenen Newton-Verfahrens annähern und das berechnete Näherungsergebnis x_n, sowie die Anzahl der benötigten Iterationen n als Tupel (x_n, n) zurückgeben.

Hinweis: Sie können die Funktion `abs` verwenden, um den Betrag einer Zahl zu berechnen.

Beispiele:

```
>>> sqrt_newton(4.0, 1e-10)
(2.0, 0)
>>> sqrt_newton(9.0, 0.5)
(3.0096153846153846, 2)
>>> sqrt_newton(9.0, 1e-10)
(3.0, 5)
>>> sqrt_newton(1337.42, 1e-10)
(36.570753341980804, 9)
```

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

Um in einem Zoo die Dosierung von Futter für alle Tiere zu organisieren, gibt es eine Datenbank in Form eines Dictionaries. In diesem sind die Namen der Tiere (`str`) die Schlüssel und die Werte ein weiteres Dictionary, das Nahrungsmittel (`str`) dem dazugehörigem Gewicht in Kilogramm (`int`) zuordnet.

(a) (10 Punkte) Die Einkaufsbeauftragte des Zoos benötigt für den monatlichen Großeinkauf eine Liste aller Gewichte für jedes Nahrungsmittel. Erstellen Sie eine Funktion `get_all_foods`, die die Datenbank `database` der Tiernahrungen entgegennimmt und eine Liste aller Nahrungsmittel mit dem benötigten Gewicht als Tupel zurückgibt. Das erste Element des Tupels ist der Name des Nahrungsmittels und das zweite Element das Gewicht. In der zurückgegebenen Liste soll kein Nahrungsmittel doppelt vorkommen, stattdessen soll das Gesamtgewicht für alle Tiere berechnet werden.

Hinweis: Mit der Funktion `items` können Sie über die Schlüssel und Werte eines Dictionaries iterieren. Außerdem können, aber müssen Sie nicht, das `defaultdict` verwenden.

```
>>> food_db = {
...     "Elephant": {"Banana": 15, "Apple": 5, "Grapes": 5},
...     "Monkey": {"Banana": 5, "Apple": 2},
...     "Giraffe": {"Banana": 10, "Apple": 5},
...     "Zebra": {"Grass": 5},
... }
>>> get_all_foods({})
[]
>>> get_all_foods(food_db)
[('Banana', 30), ('Apple', 12), ('Grapes', 5), ('Grass', 5)]
>>>
```

(b) (10 Punkte) Nach dem Einkauf werden die verschiedenen Nahrungsmittel in Schubkarren aufgeteilt und müssen zu den Gehegen gebracht werden. Um dies zu bewältigen, müssen die Zoowärterinnen wissen, mit welcher Schubkarre sie zu welchen Gehegen fahren müssen. Schreiben Sie eine Funktion `food_to_animal`, die die Datenbank `database` der Tiernahrungen entgegennimmt und ein neues Dictionary zurückgibt, in dem die Schlüssel die Nahrungsmittel sind und die Werte eine Menge von Tieren, die dieses Nahrungsmittel verzehren.

```
>>> food_to_animal(food_db)
{'Banana': {'Elephant', 'Monkey', 'Giraffe'}, 'Apple': {'Elephant',
↪  'Monkey', 'Giraffe'}, 'Grapes': {'Elephant'}, 'Grass': {'Zebra'}}
```

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

Ihre Aufgabe ist es, eine Funktion zu schreiben, die stark vereinfachte HTML-Elemente in einem String erkennen kann. Die Funktion soll die folgenden Regeln überprüfen:

1. Am Anfang steht immer ein öffnendes Tag, das mit einem `<` anfängt und mit einem `>` endet. Dazwischen müssen mindestens ein oder mehrere Buchstaben (keine Leerzeichen) stehen.
2. Auf ein öffnendes Tag folgt der Inhalt des HTML-Elements, der aus 0 oder mehreren Buchstaben und Leerzeichen bestehen kann.
3. Danach kommt immer ein schließendes Tag, das mit `</` anfängt und mit `>` endet. Dazwischen muss wie beim öffnenden Tag mindestens ein oder mehrere Buchstaben (keine Leerzeichen) stehen.
4. Der Name (Textinhalt) des öffnenden und schließenden Tags muss gleich sein.

Ein einfaches Beispiel wäre:

```
>>> element = "<p> Ein Absatz </p>"
```

Schreiben Sie eine Funktion `parse_element`, die einen beliebigen String `text` entgegennimmt und optional ein Tupel zurückgibt. Das erste Element soll der Name des öffnenden und schließenden Tags sein und das zweite Element der umschlossene Text. Wenn der String die Regeln 1-4 nicht einhält, soll `None` zurückgegeben werden.

Hinweis: Verwenden Sie die Methoden `isalpha()` und `isspace()` auf Strings, um zu überprüfen, ob ein String nur aus Buchstaben beziehungsweise Leerzeichen besteht.

```
>>> parse_element("<p> Ein Absatz </p>")
('p', ' Ein Absatz ')
>>> parse_element("<div></div>")
('div', '')
>>> parse_element("<beliebiger> Ein Absatz </text>") is None
True
>>> parse_element("< p > Ein Absatz </ p >") is None
True
>>> parse_element("<p> Ein Absatz < /p>") is None
True
```

Aufgabe 4 (Dataclasses; Punkte: 15)**.**

(a) (5 Punkte) Erstellen Sie eine Datenklasse `Animal`, die ein Tier repräsentiert und zwei Felder besitzt. Das erste Feld `name` soll den Namen (`str`) und das zweite Feld `age` das Alter (`int`) des Tieres bestimmen. Stellen Sie während der Initialisierung sicher, dass der Name ein String mit einer Länge größer als 0 ist und das Alter eine natürliche Zahl ist. Es soll außerdem eine Methode `make_sound` geben, die den Namen des Tiers, gefolgt von dem String `" makes a sound"` zurückgibt.

```
>>> animal = Animal("Fido", 5)
>>> animal.make_sound()
'Fido makes a sound'
```

(b) (5 Punkte) Erstellen Sie einen Enum `CatBreed` mit den Feldern `SHORT_HAIR`, `PERSIAN` und `SIAMESE`. Erstellen Sie dann eine Unterklasse `Cat`, die von `Animal` erbt. Die Klasse soll zusätzlich die Attribute `color` und `breed` enthalten, die die Farbe (`str`) und die Rasse (`CatBreed`) der Katze bestimmen. Stellen Sie während der Initialisierung sicher, dass die Farbe ein String mit Länge größer 0 ist.

Überschreiben Sie die Methode `make_sound` so, dass sie den Namen der Katze, gefolgt von dem Text `" meows"` zurückgibt. Wenn die Katze ein Shorthair mit der Farbe `"orange"` und dem Namen `"Garfield"` ist, soll der Text `"Garfield says: I like lasagna!"` zurückgegeben werden.

```
>>> cat = Cat("Garfield", 5, "orange", CatBreed.SHORT_HAIR)
>>> cat2 = Cat("Adam", 6, "gray", CatBreed.PERSIAN)
>>> cat.make_sound()
'Garfield says: I like lasagna!'
>>> cat2.make_sound()
'Adam meows'
```

(c) (5 Punkte) Zuletzt wollen wir Tiere anhand ihres Alters vergleichen. Implementieren Sie dafür die Dunder-Methode `__lt__` in der Klasse `Animal`, die für den Vergleich ausschließlich das Feld `age` verwendet.

```
>>> cat < cat2
True
>>> cat2 < cat
False
```

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

In der Vorlesung haben Sie Deterministische Endliche Automaten kennengelernt:

```python
@dataclass(frozen=True)
class Automaton[Q, E]:          # Zustände und Eingabealphabet
    delta: Callable[[Q, E], Q]  # Transitionsfunktion
    start: Q                    # Startzustand q0
    finals: frozenset[Q]        # Menge von Endzuständen F

    def accept(self, input):
        ...
```

Gegeben sei ein Automat $\mathcal{A}$ mit den Zuständen $Q = \{q_0, q_1, q_2, q_3, q_e\}$ über dem Alphabet $E = \{a, b, c, d\}$. Der Startzustand, die Endzustände, sowie die Transitionsfunktion sind in Abbildung 1 gegeben. Alle Kanten, die nicht explizit definiert wurden, führen in einen absorbierenden Fehlerzustand q_e.

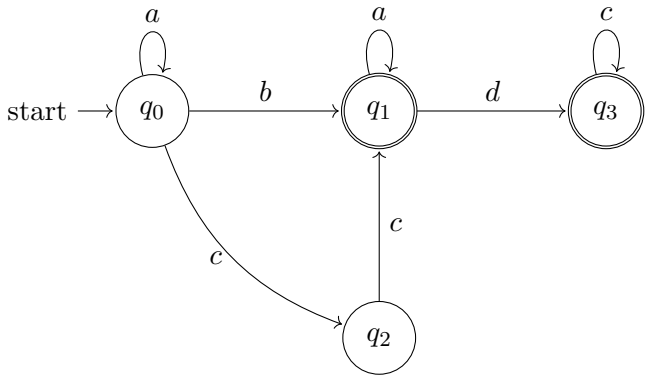

Abbildung 1: Zustandsdiagramm des Automaten $\mathcal{A}$

(a) (5 Punkte) Schreiben Sie eine Methode `accept` in der Datenklasse `Automaton`, die ein iterierbares Objekt `input` als Argument erhält. Die Methode gibt einen Wahrheitswert zurück, der angibt, ob der Automat die Eingabe akzeptiert. Der Automat akzeptiert die Eingabe, wenn er nach der Verarbeitung der gesamten Eingabe in einem Endzustand landet. Andernfalls wird die Eingabe abgelehnt.

(b) (3 Punkte) Schreiben Sie ein Enum `State` für die Zustände des Automaten $\mathcal{A}$. Benennen Sie die Felder des Enums ausschließlich mit Großbuchstaben und Zahlen. Der Startzustand q_0 soll zum Beispiel als `State.Q0` definiert werden.

Definieren Sie einen Literal-Typen `Alphabet` für das Eingabealphabet des Automaten $\mathcal{A}$.

(c) (5 Punkte) Schreiben Sie eine Funktion `delta`, die einen State `state` des Automaten $\mathcal{A}$ und ein Symbol des Eingabealphabets `input` als Argumente nimmt und den jeweiligen Folgezustand des Automaten $\mathcal{A}$ basierend auf Abbildung 1 zurückgibt. **Verwenden Sie hierzu Pattern-Matching.**

(d) (2 Punkte) Schreiben Sie anschließend eine Funktion `automaton`, die eine Instanz des Automaten $\mathcal{A}$ erstellt und diese zurückgibt.

Beispiele zu den Teilaufgaben a) bis d):

```
>>> A = automaton()
>>> A.accept(["a", "b", "a"])
True
>>> A.accept(["c", "c", "a", "d"])
True
>>> A.accept(["b", "b"])
False
```

(e) (5 Punkte) Schreiben Sie eine Funktion `reject_c`, die einen *beliebigen* Automaten `automaton` vom Typen `Automaton[State, Alphabet]` (nicht unbedingt $\mathcal{A}$!) als Argument nimmt und einen neuen Automaten erstellt und zurückgibt. Der neue Automat soll sich gleich wie `automaton` verhalten, jedoch zusätzlich alle Eingaben ablehnen, die das Symbol c enthalten (jeder Übergang mit dem Symbol c führt in den Fehlerzustand q_e). Definieren Sie dafür eine Funktion `delta_new` innerhalb der `reject_c` Funktion.

Beispiele:

```
>>> Ac = reject_c(A)
>>> Ac.accept(["a", "b", "a"])
True
>>> Ac.accept(["c", "c", "a", "d"])
False
>>> Ac.accept(["b", "b"])
False
```

Aufgabe 6 (Rekursion; Punkte: 20)**.**

Gegeben sei folgende Definition eines Binärbaumes aus der Vorlesung:

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

type BTree[T] = Optional[Node[T]]
```

Verwenden Sie diese Definition zum Lösen der folgenden Teilaufgaben.

(a) (10 Punkte) Schreiben Sie eine Funktion `sum_of_subtree`, die einen Binärbaum `tree` vom Typ `BTree[int]` als Argument entgegennimmt. Die Funktion soll den Baum so *verändern*, dass in jedem Knoten (vom Typ `Node`) das Attribut `mark` durch die Summe seines ursprünglichen Werts und der Werte aller Knoten in seinen Unterbäumen ersetzt wird. Zusätzlich gibt die Funktion die Summe aller Baumknoten zurück.

Verwenden Sie Pattern Matching und Rekursion.

Beispiele:

```
>>> t1 = None
>>> sum_of_subtree(t1)
0
>>> t1 is None
True

>>> t3 = Node(1, Node(2, Node(3), Node(4)), Node(5, Node(6)))
>>> sum_of_subtree(t3)
21
>>> t3
Node(mark=21, left=Node(mark=9, left=Node(mark=3, left=None,
↪  right=None), right=Node(mark=4, left=None, right=None)),
↪  right=Node(mark=11, left=Node(mark=6, left=None, right=None),
↪  right=None))
```

(b) (10 Punkte) Schreiben Sie eine Funktion `cut_at`, die einen beliebigen Binärbaum `tree` vom Typ `BTree[T]` sowie einen Wert `at` vom Typ `T` als Argumente erhält. Die Funktion soll einen neuen Binärbaum zurückgeben, der dem ursprünglichen Baum entspricht, allerdings werden alle Teilbäume entfernt, deren Wurzelknoten mit `at` markiert ist.

Verwenden Sie Pattern Matching und Rekursion.

Hinweis: Die Eingabe soll unverändert bleiben, stattdessen wird ein neuer Baum zurückgegeben.

Beispiele:

```
>>> cut_at(None, 42) is None
True

>>> t = Node(1, Node(2, Node(3), Node(4)), Node(3, Node(3), Node(6)))
>>> cut_at(t, 1) is None
```

```
True
>>> cut_at(t, 3)
Node(mark=1, left=Node(mark=2, left=None, right=Node(mark=4, left=None,
↪  right=None)), right=None)
```

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

Achtung: 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 `take`, die einen *Iterator* `it` und eine ganze Zahl `n` als Argumente nimmt. Die Funktion soll die ersten `n` Elemente des Iterators `it` nacheinander generieren. Falls der Iterator weniger als `n` Elemente enthält, soll er vorzeitig enden.

Beispiel:

```
>>> it = take(iter(range(10)), 5)
>>> next(it)
0
>>> list(take(it, 4))
[1, 2, 3, 4]
```

(b) (10 Punkte) Schreiben Sie eine Generator-Funktion `dependent_sum`, die einen *Iterator* `it` als Argument nimmt. Die Funktion liest jeweils eine Zahl `n` aus dem Iterator und summiert dann die nächsten `n` Zahlen, um das Ergebnis zu generieren. Falls der Iterator zu früh endet, soll die bestehende Teilsumme zurückgegeben werden. Ist der Iterator zu Beginn leer, soll eine `StopIteration` ausgelöst werden.

Beispiel:

```
>>> it = dependent_sum(iter([3, 1, 2, 3, 2, 4, 5, 42, 13]))
>>> next(it)  # 1 + 2 + 3
6
>>> next(it)  # 4 + 5
9
>>> next(it)  # 13
13
>>> next(it)  # StopIteration error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

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

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

(a) (7 Punkte) Schreiben Sie eine Funktion `words_of_length_n`, die einen String `text` und eine positive ganze Zahl `n` als Argumente nimmt. Die Funktion soll die Anzahl der Wörter in `text` zurückgeben, die genau `n` Zeichen lang sind. Wörter sind durch Leerzeichen getrennt.

Hinweis: Benutzen Sie die auf Strings definierte `split` Methode und die `len` Funktion, welche die Länger einer Liste zurück gibt.

Beispiel:

```
>>> words_of_length_n("", 42)
0
>>> text = "Hier sind vier Wörter der Länge vier"
>>> words_of_length_n(text, 0)
0
>>> words_of_length_n(text, 4)
4
```

(b) (7 Punkte) Schreiben Sie eine Funktion `update_function`, die eine einstellige Funktion `f`, einen Eingabewert `inp` und einen Ausgabewert `out` als Argumente nimmt. Die Funktion soll eine neue Funktion zurückgeben, die sich wie `f` verhält, außer dass sie für `inp` den Wert `out` liefert.

Beispiel:

```
>>> inc = lambda x: x + 1
>>> inc(0)
1
>>> upd_inc = update_function(inc, 0, 42)
>>> upd_inc(0)
42
>>> upd_inc(1)
2
```

(c) (6 Punkte) Schreiben Sie eine Funktion `case_of`, die einen Wert `x` von beliebigem Typ, eine Liste von Mustern `patterns`, eine Liste von Funktionen `rhs` (rechte Seiten) und eine Standardfunktion `default` als Argumente nimmt. Sie können davon ausgehen, dass die Listen `patterns` und `rhs` gleich lang sind. Falls `x` in `patterns` enthalten ist, soll die entsprechende Funktion aus `rhs` mit `x` aufgerufen werden. Andernfalls soll `default` von `x` zurückgegeben werden.

Hinweis: Benutzen Sie die auf Listen definierte `index` Methode. Die Methode gibt, aufgerufen für einen Wert, der in der Liste vorkommen muss, den Listenindex des Wertes in der Liste zurück:

```
>>> lst = [42]
>>> lst.index(42)
0
>>> lst.index(17)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 17 is not in list
```

Beispiel:

```
>>> patterns = [0, 1, 42]
>>> rhs = [
...     lambda x: f"Ich habe {x} Verständnis für diese Aufgaben!",
...     lambda x: f"{x} ist mehr als 0!",
...     lambda x: f"Der Sinn des Lebens ist {x}!",
... ]
>>> default = lambda x: f"Was soll ich denn mit einer {x}?"
>>> case_of(42, patterns, rhs, default)
'Der Sinn des Lebens ist 42!'
>>> case_of(17, patterns, rhs, default)
'Was soll ich denn mit einer 17?'
```