Autor:
Jakob Haverkamp

Einführung in die Programmierung
WS 2025/2026
Prof. Dr. Peter Thiemann

Klausur für mein Tutorat

04. Februar 2026

💡 **Besprechung der Klausur am 13.02 um 10:00 in Geb. 101, SR 01-016/18** 💡

Disclaimer: Ich garantiere weder für Korrektheit der Lösungen noch dafür, dass die Klausur ähnlich wie diese wird. Bei Fragen: `jh1444@email.uni-freiburg.de`

Diese Klausur ist dazu da euch zu helfen und ich hoffe ihr könnt damit üben, die Lösungen werde ich im Anschluss an das Tutorat veröffentlichen. Für die Erstellung einzelner Aufgaben habe ich aus Zeitgründen KI zu Hilfe genommen. Danke an Lutz und Jannis für das korrekturlesen, testen und teilweises schreiben der Musterlösung :)

- 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. `ex3_strings.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.

Es sind maximal 150 Punkte zu erreichen.

Aufgabe 1 (Warm-Up; 20 Punkte)
In dieser Aufgabe sollen 6 kurze Aufgaben gelöst werden:

(a) (3 Punkte) Definieren Sie eine Funktion `my_all()`, die eine Liste von Wahrheitswerten xs als Argument nimmt und zurückgibt, ob alle Werte in xs wahr sind. **Verwenden sie nicht die `all()`-Funktion aus der Standard-Bibliothek.**

```
>>> my_all([True, False])
False
>>> my_all([])
True
```

(b) (4 Punkte) Definieren Sie eine Funktion `einkaufsliste()`, die die BenutzerIn so lange nach Einkaufsartikeln fragt, bis diese „Fertig“ schreibt. Alle Artikel, die bis dahin gekauft werden sollen, sollen als Set von Zeichenketten zurückgegeben werden.

Ein Aufruf der Funktion mit den Eingaben Apfel, Schokolade und Fertig soll exakt wie folgt aussehen:

```
>>> einkaufsliste()
>>> Einkauf: Apfel
>>> Einkauf: Schokolade
>>> Einkauf: Fertig
{"Apfel", "Schokolade"}
```

(c) (6 Punkte) Definieren Sie zwei Funktionen `mult_rec()` und `mult_it()`, welche die Multiplikation von zwei **positiven Ganzzahlen** `a` und `b` einmal rekursiv und einmal iterativ implementiert. Dafür dürfen sie nicht den * oder /-Operator benutzen.

```
>>> mult_rec(5, 4)
20
>>> mult_it(5, 4)
20
>>> mult_it(5, 5) == mult_rec(5, 5) == 5*5
True
```

(d) (2 Punkte) Gegeben sei die rekursive Definition eines Binärbaumes in Python:

```
@dataclass
class Node[T]:
  mark: T
  left: "Tree[T]"
  right: "Tree[T]"

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

Im Template finden Sie diese Definition, ändern Sie diese so ab, dass jeder Knoten des Baumes sowohl ein `mark`, als auch ein weiteres Attribut `number` hat, welches

der Nummerierung der Knoten dient und einen dafür geeigneten Datentyp hat. Zusätzlich soll jeder Baum ein drittes Kind `middle` neben `left` und `right` haben.

(e) (5 Punkte) Definieren sie die Funktion `apply_n_times()` welche eine einstellige Funktion ohne Rückgabe **f**, ein Argument für diese Funktion **arg** und eine Ganzzahl **n** nimmt und die Funktion **n**-mal ausführt. Ein Beispiel sieht so aus:

```
>>> apply_n_times(print, "Hallo Welt!", 4)
Hallo Welt!
Hallo Welt!
Hallo Welt!
Hallo Welt!
```

Aufgabe 2 (Dictionaries und Sets; 15 Punkte)
Eine Getränkekarte (dict) ordnet Cocktails (str) die Mengen ihrer Zutaten (str) und ihren Preis (float) zu. Zum Beispiel:

```
drinks = {
        "Daiquiri": ({"Rum", "Limette", "Zucker"}, 7.90),
        "Mojito": ({"Rum", "Limette", "Zucker", "Minze", "Soda"}, 7.50),
        "Whiskey Sour": ({"Whiskey", "Zitrone", "Zucker"}, 8.90),
        "Tequila Sour": ({"Tequila", "Zitrone", "Zucker"}, 6.50),
        "Moscow Mule": ({"Vodka", "Limette", "Ginger ale"}, 7.00),
}
```

(a) (4 Punkte) Schreiben sie eine Funktion `buyable()`, welche eine Getränkekarte `drinks` und eine Gleitkommazahl `money` als Argumente nimmt und die Menge aller damit kaufbaren Cocktails zurückgibt.

```
>>> buyable(drinks, 5.00)
set()
>>> buyable(drinks, 7)
{"Tequila Sour", "Moscow Mule"}
```

(b) (6 Punkte) Damit Allergiker schnell wissen, welche Cocktails sie trinken können, wollen wir uns eine Allergikerkarte machen, in der jede Zutat ein Schlüssel ist.

Schreiben sie eine Funktion `invert()`, welche eine Getränkekarte `drinks` als Argument nimmt und die ein neues Dictionary zurückgibt, welches jeder Zutat die Menge an mischbaren Cocktails zurückgibt

```
>>> invert(drinks)
{"Limette": {"Daiquiri", "Mojito", "Moscow Mule"}, "Zitrone": ... usw}
```

(c) (5 Punkte) Benutzen sie die Funktionen aus den vorheringen beiden Aufgaben um einer AllergikerIn die Bestellung zu erkleichtern. Dafür sollen sie die Funktion `buyable_allergic` schreiben. Diese Funktion soll unsere originale Karte `drinks`, eine Gleitkommazahl `money` und eine Menge an Zutaten `allergic` nehmen und die Cocktails zurückgeben, welche die AllergikerIn trinken und kaufen kann.

```
>>> buyable_allergic(drinks, 5.00, {"Limette", "Kokos"})
set()
>>> buyable_allergic(drinks, 7.00, {"Limette", "Kokos"})
{"Tequila Sour"}
```

Aufgabe 3 (Strings; 20 Punkte)

(a) (5 Punkte) Bei der Cäsar-Verschlüsselung werden alle Buchstaben im Alphabet um eine bestimmte Zahl verschoben. Für einen Schlüssel `key = 3` würde aus einem `"a"` z. B. ein `"d"`

Schreiben Sie die Funktionen `encode`, die eine Zeichenkette `msg` und eine Ganzzahl `key` nimmt und den Text mit der oben gezeigten Methode verschlüsseln.

Hinweis: `ord()` verwandelt einen Buchstaben in die dazugehörige Ganzzahl des Unicode code, `chr()` wandelt die Zahl zurück. Eine Unicode-Tabelle sieht so aus:

```
| letter | unicode |
| ------ | ------- |
|   a    |   97    |
|   b    |   98    |
|   c    |   99    |
|   d    |   100   |
|  ...   |   ...   |
```

```
>>> encode("abc", 3)
"def"
>>> encode("Hallo Welt!", 5)
"Mfqqt%\jqy&"
```

(b) (15 Punkte) Schreiben Sie eine Funktion `validate_email`, die einen String `email` als Argument nimmt und überprüft, ob dieser eine gültige einfache E-Mail-Adresse darstellt. Eine gültige E-Mail-Adresse muss folgende Bedingungen erfüllen:

- Sie enthält genau ein @-Zeichen und einen Punkt.
- Vor dem @ steht eine nichtleere Zeichenkette aus Buchstaben und Zahlen
- Nach dem @ folgt mindestens ein Buchstabe, dann genau ein Punkt, dann mindestens zwei Buchstaben (die Domain)
- Die E-Mail-Adresse darf nur Buchstaben (a-z, A-Z) und Zahlen (0-9) enthalten.
- Die Funktion soll True zurückgeben, wenn alle Bedingungen erfüllt sind, sonst False

Hinweis: Verwenden Sie die String-Methoden `isalpha()`, `isalnum()` oder `count()`.

```
>>> validate_email("user@example.com")
True
>>> validate_email("test.user_123@domain.com")
False
>>> validate_email("invalid@@email.com")
False
```

Aufgabe 4 (Datenklassen; 15 Punkte)
Diese Aufgabe ist wahrscheinlich einfacher als die tatsächliche Dataclasses Aufgabe.

(a) (5 Punkte) Erstellen Sie eine Datenklasse Pokemon, die ein Namen `name`, eine ganzzahlige Anzahl Lebenspunkte `hp`, eine ganzzahlige Angriffsstärke `attack` und eine Klasse `pokemon_type` hat, die entweder Feuer, Wasser, Luft oder Erde ist.

Stellen Sie durch `assert`-Anweisungen sicher, dass:
- Der Name nicht leer ist
- Die Lebenspunkte größer als 0 sind
- Die Angriffsstärke größer als 0 ist
- Der pokemon_type nicht leer ist

Implementieren Sie eine Methode `take_damage`, die einen ganzzahligen Schaden damage als Argument nimmt und die Lebenspunkte entsprechend reduziert (mindestens auf 0).

Implementieren Sie eine Methode `is_alive`, die `True` zurückgibt, wenn das Pokemon noch Lebenspunkte hat, sonst False.

(b) (5 Punkte) Erstellen Sie eine Datenklasse `WaterPokemon`, die von Pokemon erbt. Wasser-Pokemon haben eine zusätzliche Fähigkeit: Sie können sich selbst heilen.

Der `pokemon_type` soll automatisch auf `Wasser` gesetzt werden und muss nicht beim Initialisieren angegeben werden.

Implementieren Sie eine Methode `heal`, die eine ganzzahlige Heilmenge `amount` als Argument nimmt und die Lebenspunkte um diesen Betrag erhöht. Die Lebenspunkte dürfen dabei einen maximalen Wert `max_hp` nicht überschreiten. Fügen Sie dafür ein zusätzliches Attribut `max_hp` hinzu, das bei der Initialisierung auf den Anfangswert von `hp` gesetzt wird.

Verwenden Sie `super()`, um unnötige Code-Duplikation zu vermeiden.

(c) (5 Punkte) Erstellen Sie eine Datenklasse `FirePokemon`, die ebenfalls von Pokemon erbt. `Feuer-Pokemon` haben eine stärkere Angriffskraft. Der `pokemon_type` soll automatisch auf `Feuer` gesetzt werden.

Implementieren Sie eine Methode `fire_attack`, die ein anderes Pokemon `opponent` als Argument nimmt. Ein Feuer-Angriff verursacht 1.5-fachen Schaden (abgerundet auf ganze Zahlen). Die Methode soll dem Gegner entsprechend Schaden zufügen.

Hinweis: Verwenden Sie die Funktion `int()`, um auf ganze Zahlen abzurunden

Aufgabe 5 (Endrekursion; 15 Punkte)

(a) (5 Punkte) Schreiben Sie eine **rekursive, aber nicht endrekursive** Funktion `sum_list_rec`, die eine Liste von ganzen Zahlen `lst` als Argument nimmt und die Summe aller Elemente berechnet.

(b) (5 Punkte) Schreiben Sie eine endrekursive Funktion `sum_list_tail`, die sich wie `sum_list_rec` verhält, aber endrekursiv implementiert ist. Verwenden Sie hierzu das in der Vorlesung gezeigte Verfahren mit einem Akkumulator acc als zusätzlichem Argument.

(c) (5 Punkte) Schreiben Sie eine nicht-rekursive Funktion `sum_list_iter`, die sich wie `sum_list_rec` verhält, aber endrekursiv implementiert ist. Verwenden Sie hierzu das in der Vorlesung gezeigte Verfahren mit einem Akkumulator acc als zusätzlichem Argument.

Beispiel:

```
>>> l = [1, 5, 3, 4]
>>> sum_list_rec(l)
13
>>> sum_list_rec(l) = sum_list_tail(l) = sum_list_iter(l)
True
```

Aufgabe 6 (Bäume und Rekursion; 25 Punkte)
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]]
```

(a) (5 Punkte) Schreiben Sie eine Funktion `count_leaves`, die einen Binärbaum `tree` als Argument nimmt und die Anzahl seiner Blätter zurückgibt. Ein Blatt ist ein Knoten ohne Kinder. **Verwenden Sie Pattern Matching und Rekursion.** Beispiel:

```
>>> count_leaves(None)
0
>>> count_leaves(Node(1))
1
>>> count_leaves(Node("zero", Node("one"), Node("two")))
2
```

(b) (10 Punkte) Schreiben Sie eine Funktion `tree_map`, die eine einstellige Funktion `f` und einen Binärbaum `tree` als Argumente nimmt. Die Funktion soll einen neuen Baum zurückgeben, bei dem `f` auf alle Markierungen angewendet wurde. **Verwenden Sie Pattern Matching und Rekursion.**

```
>>> t2 = tree_map(lambda x: x * 2, Node(1, Node(2), Node(3)))
>>> t2 == Node(2, Node(4), Node(6))
True
```

(c) (8 Punkte) Für die folgende Aufgabe betrachten wir Suchbäume, also Bäume, bei denen das Linke Kind immer kleiner und das rechte Kind immer größer ist als der Wert des Knotens.

Schreiben Sie die Funktion `tree_to_list`, welche einen Suchbaum mit Ganzzahlen als Inhalt `tree` nimmt und ihn so in eine Liste unwandelt, dass diese Liste strikt aufsteigend geordnet ist.

(d) (2 Punkte) Schreiben sie die Funktion `test_ordered`, welche eine Liste von Ganzzahlen `xs` nimmt und überprüft, ob diese strikt aufsteigend geordnet ist. Wenn Sie wollen können Sie die Ausgabe der `tree_to_list`-Funktion mit `test_ordered` überprüfen.

```
>>> test_ordered([0, 1, 2])
True
```

Aufgabe 7 (Generatoren; 20 Punkte)

(a) (5 Punkte) Schreiben Sie eine Generator-Funktion `take`, die ein iterierbares Objekt `xs` und eine ganze Zahln als Argumente nimmt. Der Generator soll die ersten `n` Elemente von `xs` generieren.

Beispiel:

```
>>> list(take(range(10), 3))
[0, 1, 2]
>>> list(take("Hello World", 5))
['H', 'e', 'l', 'l', 'o']
>>> list(take([1, 2], 5))
[1, 2]
```

(b) (5 Punkte) Schreiben Sie eine Generator-Funktion `interleave`, die zwei iterierbare Objekte `xs` und `ys` als Argumente nimmt und abwechselnd Elemente aus beiden generiert. Wenn ein Iterator erschöpft ist, sollen die restlichen Elemente des anderen Iterators generiert werden

Beispiel:

```
>>> list(interleave([1, 2, 3], ['a', 'b', 'c']))
[1, 'a', 2, 'b', 3, 'c']
>>> list(interleave([1, 2], ['a', 'b', 'c', 'd']))
[1, 'a', 2, 'b', 'c', 'd']
>>> list(interleave([1, 2, 3], []))
[1, 2, 3]
```

(c) (10 Punkte) Schreiben Sie eine Generator-Funktion `primes`, welche unbeschränkt ist und Primzahlen generiert. Eine Primzahl ist jede positive Ganzzahl, die nur durch 1 und sich selbst teilbar ist.

Beispiel:

```
>>> prim_gen = primes()
>>> [next(prim_gen) for _ in range(7)]
[1, 2, 3, 5, 7, 11, 13]
```

Aufgabe 8 (Funktionale Programmierung; 20 Punkte)

(a) (5 Punkte) Schreiben Sie eine Dekorator-Funktion `block`, welche eine Funktion nicht ausführt, sondern beim Aufruf der dekorierten Funktion `"Diese Funktion wurde blockiert!"` ausgibt.

Beispiel:

```
@block
def test():
    print("Ausgeführt")
```

```
>>> test()
Diese Funktion wurde blockiert!
```

(b) (5 Punkte) Schreiben Sie eine Funktion `filter_dict`, welche eine zweistellige Funktion `f` mit den Argumenten `key` und `value` und ein dictionary `dic` nimmt, und ein neues Dictionary zurückgibt, in der alle Werte entfernt sind, für die die Funktion `f` `False` zurückgibt.

Hinweis: Ihre Funktion aus einem einzigen `return`-Ausdruck bestehen.

Beispiel:

```
d = {"a": 1, "b": 2, "c": 3, "d": 4}
>>> filter_dict(lambda x, y: y % 2 == 0, d)
{"b": 2, "d": 4}
```

(c) (5 Punkte) Schreiben Sie eine Funktion `transpose`, die eine Matrix (Liste von Listen) matrix als Argument nimmt und die transponierte Matrix zurückgibt.

Hinweis: Verwenden Sie eine verschachtelte List-Comprehension.
Hinweis: Ihre Funktion aus einem einzigen `return`-Ausdruck bestehen.

```
>>> transpose([[1, 2, 3], [4, 5, 6]])
[[1, 4], [2, 5], [3, 6]]
>>> transpose([[1], [2], [3]])
[[1, 2, 3]]
```