Informatik 1
Einführung in die Programmierung
WS 2022

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

- Für die Bearbeitung der Aufgaben haben Sie **120 Minuten** Zeit.
- Es sind **keine Hilfsmittel** wie Skripte, Bücher, Notizen oder Taschenrechner erlaubt. Desweiteren 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 Rückgabe- und Parameter-Typen Ihrer Funktion anzugeben. Fehlende Typannotationen führen zu Punktabzug.
- Bearbeiten Sie die einzelnen Aufgaben in den vorgegebenen **Templates**, z.B. `ex1_sequences.py`. **Falsch benannte Funktionen** werden nicht bewertet. **Neu erstellte Dateien** werden nicht bewertet.
- Bei Bedarf können Sie **weitere Module** aus der Standardbibliothek importieren. Zum Lösen der Aufgaben ist dies aber nicht notwendig.

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

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

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

$$f(n) = \begin{cases} \frac{n}{2} & n \text{ ist teilbar durch } 2 \\ 9n + 3 & \text{sonst} \end{cases}$$

Wir sind daran interessiert, wie oft die Funktion f auf sich selbst angewendet werden muss bis ein bestimmtes Limit a überschritten wird. Schreiben Sie dafür eine Funktion `count_iterations`, welche eine natürliche Zahl $n > 0$ und ein Limit $a > 0$ als Argument nimmt und zurück gibt, nach wie vielen Anwendungen von f eine Zahl $x > a$ zurückgegeben wird. Beispiel:

```
>>> count_iterations(3, 100)
3
```

Die Begründung hierfür ist folgende: $f(3) = 30, f(30) = 15, f(15) = 138$ und $138 > 100$

Aufgabe 2 (Dictionaries und Sets; Punkte: 10)**.**

Die Verwaltung der Universität möchte jeder Lehrperson die Menge derjenigen Studierenden zuordnen, die sie dieses Semester unterrichtet.
Es ist bereits ein Dictionary vorhanden, welches jeder Lehrperson die Menge ihrer Vorlesungen zuordnet (`prof_to_courses`), und ein Dictionary, welches jeder Vorlesung, die Menge der teilnehmenden Studierenden zuordnet (`course_to_students`).
Helfen Sie der Universitätsverwaltung und schreiben Sie eine Funktion `compose`, die ein Dictionary `prof_to_courses` und ein Dictionary `course_to_students` als Argumente nimmt und ein Dictionary zurückgibt, welches jeder Lehrperson die Menge derjenigen Studierenden zuordnet, die sie unterrichtet.
Lehrpersonen, Vorlesungen und Studierende werden jeweils durch Strings repräsentiert.
Beispiel:

```
>>> prof_to_courses = {
>>>     "Prof A": {"Course A", "Course B"},
>>>     "Prof B": {"Course B", "Course C"},
>>> }
>>> course_to_students = {
>>>     "Course A": {"Student A", "Student B"},
>>>     "Course B": {"Student B", "Student C"},
>>>     "Course C": {"Student D", "Student E"},
>>> }
>>> compose(prof_to_courses, course_to_students)
{
    "Prof A": {"Student A", "Student B", "Student C"},
    "Prof B": {"Student B", "Student C", "Student D", "Student E"},
}
```

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

Schreiben Sie eine Funktion `is_list_of_int`, die einen String `x` als Argument nimmt und überprüft, ob der String eine gültige Liste von ganzen Zahlen darstellt. Die Gültigkeit dieser Liste ist wie folgt definiert:

- Der String beginnt mit einer offenen eckigen Klammer "`[`" und endet mit einer geschlossenen eckigen Klammer "`]`".
- Eine ganze Zahl besteht aus einem oder mehreren Ziffern "`0`" bis "`9`" und beginnt mit maximal einem Minus "`-`".
- Zwischen jeweils zwei ganzen Zahlen muss ein Komma "`,`" stehen, ansonsten dürfen keine Kommata enthalten sein.
- Es dürfen keine anderen Zeichen (insbesondere auch keine Leerzeichen) enthalten sein.
- In der Liste muss mindestens eine Zahl enthalten sein.
- Die Funktion soll für jeden möglichen String einen Wahrheitswert zurückgeben, also insbesondere für keinen String einen Fehler auswerfen.

Beispiel:

```
>>> is_list_of_int("[100,-44,0]")
True
>>> is_list_of_int("[123,456,]")
False
```

Hinweis: Die Verwendung von `isdigit` wird empfohlen.

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

(a) (6 Punkte) Ein `Name` besteht aus einem einem Nachnamen (`surname`) und einem Vornamen (`prename`). `Name`n werden üblicherweise primär nach dem Nachnamen sortiert und nur wenn die Nachnamen gleich sind nach dem Vornamen.

Definieren Sie eine entsprechende Datenklasse `Name`, die eine magische `__lt__`-Methode besitzt, die zwei `Name`n primär nach dem Nachnamen vergleicht.

(b) (7 Punkte) In einem Fenster können verschiedene Ereignisse (Events) auftreten, die durch Tastatur und Maus ausgelöst werden. Ein Event ist entweder

- ein `MouseClick`, der aus einer ganzzahligen `x`- und `y`-Position besteht; oder
- ein `KeyPress`, der aus der gedrückten Taste `key` besteht, die wir zur Einfachheit als String repräsentieren.

Definieren Sie die zugehörigen Datenklassen für Events und eine Funktion `log_event`, die ein Event als Argument nimmt und wie folgt auf der Kommandozeile ausgibt:

- Ein Mausklick an der Position `(3, 2)` soll durch folgenden Text ausgegeben werden:

  ```
  Mouse clicked at (3, 2).
  ```

- Ein Druck der Taste `'a'` soll durch folgenden Text ausgegeben werden:

  ```
  Key 'a' pressed.
  ```

Verwenden Sie dabei *keine* Vererbung, sondern Union-Types und Pattern Matching, um zwischen den verschiedenen Events zu unterscheiden.

(c) (7 Punkte) In einem Spiel gibt es verschiedene Spielobjekte (`GameObject`s).

Alle Spielobjekte haben eine `x`- und `y`-Position und eine `move`-Methode, die das Spielobjekt relativ zu seiner aktuellen Position verschiebt. Hierzu nimmt die `move`-Methode zwei ganzzahlige Argumente `dx` und `dy`.

Ein Spieler (`Player`) ist ein spezielles Spielobjekt, das neben seiner Position noch ein ganzzahliges Attribut `health` besitzt.

Die `x`- und `y`-Positionen von jedem Spielobjekt sind nur gültig, wenn sie nicht negativ sind. Ebenso darf das `health`-Attribut des Spielers nicht negativ sein.

Implementieren Sie die `GameObject`- und `Player`-Klassen inklusive Methoden und verwenden Sie Vererbung, um unnötig doppelten Code zu vermeiden. Stellen Sie durch `assert`-Anweisungen sicher, dass die oben genannten Invarianten eingehalten werden und verwenden Sie `super`, um auch hier unnötig doppelten Code zu vermeiden.

Der Aufgabenteil darf sowohl mit Datenklassen als auch mit normalen (undekorierten) Klassen gelöst werden.

Aufgabe 5 (Tests; Punkte: 10)**.**

Schreiben Sie `pytest`-kompatible Unittests für die folgende Funktion. Dabei soll jede `return`-Anweisung durch genau eine Testfunktion abgedeckt werden.

```
def is_strong(pw: str) -> bool:
    int_count = 0
    upper_count = 0
    used_specials = []

    for c in pw:
        if c.isupper():
            upper_count += 1
        elif c.isdigit():
            int_count += 1
        elif not c.islower():
            used_specials += [c]

    if int_count < 1:
        return False
    elif int_count <= 3:
        if len(used_specials) > 0:
            used_specials.pop(0)
        else:
            return False

    if upper_count <= 2:
        if len(used_specials) > 0:
            used_specials.pop(0)
        else:
            return False
    return True
```

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

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

```
@dataclass
class Node:
    mark: str
    left: Optional['Node']
    right: Optional['Node']

Tree = Optional[Node]  # Trees can be empty.
```

Ein *Pfad* beschreibt wie ein Knoten ausgehend von der Wurzel eines Baumes erreicht werden kann. Wir stellen einen Pfad durch einen String dar, der aus den Zeichen `'l'` und `'r'` besteht. Das Zeichen `'l'` steht für den linken Teilbaum; das Zeichen `'r'` für den rechten Teilbaum.
Beispiel: Man betrachte den Baum

```
example = Node("n0",
               Node("n1", None, None),
               Node("n2",
                   Node("n3", None, None),
                   Node("n4", None, None)
               )
          )
```

- Der leere Pfad `""` beschreibt den Wurzelknoten `n0`.
- Der Pfad `"r"` beschreibt den rechten Teilbaum `n2` des Wurzelknotens.
- Der Pfad `"rl"` beschreibt den linken Teilbaum `n3` des rechten Teilbaums des Wurzelknotens.

(a) (8 Punkte) Schreiben Sie eine Funktion `mark_at_path`, die einen Baum `tree` und einen Pfad `path` als Argumente nimmt und die Markierung desjenigen Knotens zurückgibt, der durch den Pfad `path` beschrieben wird. Beschreibt `path` keinen Knoten des Baumes, so soll `None` zurückgegeben werden. Beispiel:

```
>>> mark_at_path(example, "")
"n0"
>>> mark_at_path(example, "rl")
"n3"
>>> mark_at_path(example, "ll")
None
```

(b) (7 Punkte) Schreiben Sie eine Funktion `paths`, die einen Baum `tree` als Argument nimmt und die Liste aller Pfade von `tree` zurückgibt. Der Baum soll in *Pre-Order*-Reihenfolge durchlaufen werden. Beispiel:

```
>>> paths(example)
["", "l", "r", "rl", "rr"]
>>> paths(None)
[]
```

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

Der Import von zusätzlichen Modulen ist in dieser Aufgabe **verboten**.
Vermeiden Sie unnötigen Speicherverbrauch: Funktionen, die iterierbare Objekte (Iterables) als Argument nehmen, dürfen diese nicht unnötigerweise in eine Liste umwandeln.
In den folgenden Teilaufgaben müssen Sie *keine* Typannotationen angeben.

(a) (6 Punkte) Schreiben Sie eine Generator-Funktion `my_chain`, die zwei iterierbare Objekte `xs` und `ys` als Argumente nimmt und zuerst die Elemente aus `xs` und dann die Elemente aus `ys` zurückgibt.

Beispiel:

```
>>> list(my_chain(range(5), range(2)))
[0, 1, 2, 3, 4, 0, 1]
```

(b) (7 Punkte) Schreiben Sie eine Generator-Funktion `dup`, die eine ganze Zahl `n` und ein iterierbares Objekt `xs` als Argumente nimmt und sich wie `xs` verhält, aber jedes Element `n`-mal nacheinander generiert.

Beispiel:

```
>>> list(dup(3, range(1, 4)))
[1, 1, 1, 2, 2, 2, 3, 3, 3]
```

(c) (7 Punkte) Schreiben Sie eine Generator-Funktion `compare`, die zwei iterierbare Objekte `xs` und `ys` von ganzen Zahlen als Argumente nimmt und für das nächste Element aus `xs` und `ys` jeweils das Größere der beiden Elemente zurückgibt. Ist einer der Generatoren erschöpft, soll die Ausgabe beendet werden.

Beispiel:

```
>>> list(compare(range(6), range(-3, 7, 3)))
[0, 1, 3, 6]
```

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

In den folgenden Teilaufgaben müssen Sie *keine* Typannotationen angeben.
Implementieren Sie die Funktionen aus folgenden Teilaufgaben im funktionalen Stil - also entweder mit einem Rumpf, der aus genau einer `return`-Anweisung besteht, oder durch ein Lambda, das einer Variable zugewiesen wird.

(a) (5 Punkte) Schreiben Sie eine Funktion `times_2`, die eine einstellige Funktion `f` als Argument nimmt und eine einstellige Funktion zurückgibt, die sich wie `f` verhält, aber das Ergebnis von `f` mit zwei multipliziert.

Beispiel:

```
>>> f = lambda x: x + 1
>>> g = times_2(f)
>>> g(3)
8                        # because (3 + 1) * 2 = 8
```

(b) (5 Punkte) Eine Matrix kann durch eine Liste von Listen von Zahlen dargestellt werden. Beispiel:

```
example = [ [ 1, 2, 3 ]
          , [ 4, 5, 6 ] ]
```

Schreiben Sie eine Funktion `map_matrix`, die eine Funktion `f` und eine Matrix `m` als Argument nimmt und eine Matrix zurückgibt, in der die Funktion `f` auf jede Zelle der Matrix `m` angewandt wurde. Verwenden Sie hierzu eine verschachtelte List-Comprehension.

Beispiel:

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

(c) (5 Punkte) Schreiben Sie eine Funktion `map_matrix_2`, die sich wie die Funktion `map_matrix` aus dem vorherigen Aufgabenteil verhält, aber verwenden Sie bei der Implementierung statt List-Comprehensions *mehrere* Aufrufe der `map`- und `list`-Funktionen und einen `lambda`-Ausdruck.