Informatik 1
Einführung in die Programmierung
WS 2020

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.
- Falls Sie **mehrere Lösungsansätze** einer Aufgabe erarbeiten, markieren Sie deutlich, welcher gewertet werden soll.
- Verwenden Sie **Typannotationen** um die Rückgabe- und Parameter-Typen Ihrer Funktion anzugeben. Fehlende Typannotationen führen zu einem geringen Punktabzug.
- Bearbeiten Sie die einzelnen Aufgaben in den vorgegebenen **Templates**, z.B. `ex1_sequences.py`. Falsch benannte Funktionen 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 $f : \mathbb{N} \to \mathbb{N}$,

$$f(n) = \begin{cases} n+4 & n \text{ ist teilbar durch } 3 \\ \frac{n}{2} & n \text{ ist nicht durch 3, aber durch 4 teilbar} \\ n-1 & n \text{ ist weder durch 3 noch 4 teilbar} \end{cases}$$

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

```
>>> count_iterations(5)
4
```

Die Begründung hierfür ist folgende: $f(5) = 4, f(4) = 2, f(2) = 1, f(1) = 0$

Aufgabe 2 (Dict; Punkte: 10)**.**

Betrachten Sie folgendes Dictionary, welches Studenten ihre erreichten Klausurpunktzahlen zuordnet:

```
points = {"Line": 9, "Daniel": 12, "Charlotte": 15, "Frank": 30}
```

Schreiben Sie eine Funktion `cluster_by_points`, die solch ein Dictionary `points` als Argument nimmt und ein Dictionary zurückgibt, das Studenten mit Punktzahlen innerhalb eines ähnlichen Bereichs zusammenfasst.
Um dies zu erreichen, soll die Punktzahl von jedem Studenten auf das nächst niedrigere Vielfache von zehn gerundet werden. So wird z.B. eine Punktzahl von 9 auf 0 gerundet, während 15 Punkte auf 10 gerundet werden.
Diese gerundeten Punkte sollen nun die Schlüssel des neuen Dictionary sein, während die Studenten selbst in den zugehörigen Werten gelistet sind. Beispiel:

```
>>> cluster_by_points(points)
{0: ['Line'], 10: ['Daniel', 'Charlotte'], 30: ['Frank']}
```

Das Dictionary soll keine Einträge enthalten deren Wert eine leere Liste ist.

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

Schreiben Sie eine Funktion `is_strong` die prüft ob ein gegebenes Passwort (`pw: str`) **stark** ist oder nicht. Ein Passwort gilt als **stark** wenn folgende Punkte erfüllt sind:

1) Das Passwort besteht aus mindestens acht Zeichen.
2) Es muss mindestens eine Ziffer vorkommen.
3) Sind im Passwort nur drei oder weniger Ziffern, dann muss es mindestens ein Sonderzeichen geben.
4) Wenn die Anzahl der Großbuchstaben unter drei liegt, dann muss ebenfalls ein Sonderzeichen existieren.
5) Sollte sowohl wegen Punkt 3) als auch wegen Punkt 4) ein Sonderzeichen im Passwort vorkommen, dann müssen insgesamt mindestens 2 Sonderzeichen vorhanden sein (wenn z.B. zwei `'?'` im Passwort vorkommen ist das ausreichend).

Die Funktion soll `True` oder `False` zurückgeben, je nachdem ob das Passwort als **stark** gilt oder nicht. Zur Einfachheit gehen wir davon aus dass das Passwort nur aus den Buchstaben `a-z` bzw. `A-Z`, den Ziffern `0-9` und den Sonderzeichen `'!', '?', '+'` und `'*'` besteht.
Hinweis: Die Verwendung von `isupper` und `isdigit` wird empfohlen.

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

In dieser Aufgabe soll ein Fußballtabelleneintrag mittels Datenklassen repräsentiert werden.

(a) (6 Punkte) Schreiben Sie eine Datenklasse `Ranking` mit folgenden Attributen:

- `club: str`, der Name des Vereins
- `wins: int`, die Anzahl der gewonnen Spiele
- `draws: int`, die Anzahl der Unentschieden
- `losses: int`, die Anzahl der verlorenen Spiele
- `goals_achieved: int`, Tore die der Verein erzielt hat
- `goals_conceded: int`, Tore die der Verein erlitten hat

(b) (6 Punkte) Implementieren Sie eine Methode `table_entry` von `Ranking` welche einen String, der einen Tabelleneintrag repräsentiert, zurück gibt. Ein Tabelleneintrag besteht aus dem Vereinsnamen, der Anzahl aller Spiele, den Siegen, den Unentschieden, den Niederlagen, dem Torverhältnis, der Tordifferenz und den Punkten. Die Reihenfolge soll eben diese sein. Das Torverhältnis zeigt sowohl die erzielten wie auch die erlittenen Tore an. Diese beiden Werte werden durch einen Doppelpunkt getrennt. Jeder Sieg gibt drei, jedes Unentschieden einen Punkt. Niederlagen verändern die Punkte nicht. Beispiel des Ausgabestrings:

```
>>> Ranking("FC H.", 6, 2, 2, 23, 14).table_entry()
FC H. 10 6 2 2 23:14 9 20
```

(c) (8 Punkte) Implementieren Sie die Vergleichsmethode `<` (`__lt__`) für zwei Vereine. Stellen Sie sicher, dass hier auch wirklich 2 Objekte vom Typ `Ranking` verglichen werden. Die Vergleichsmethode vergleicht primär die Punkte der beiden Vereine. Sollten diese gleich sein, wird die Tordifferenz verglichen. Ist auch die Tordifferenz identisch, werden die erzielten Tore verglichen.

```
>>> r1 = Ranking("FC H.", 6, 2, 2, 23, 14)
>>> r2 = Ranking("FC U.", 5, 3, 2, 20, 15)
>>> r2 < r1
True
>>> r1 < r1
False
```

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_prime(n: int) -> bool:
  if n == 2:
      return True
  elif n < 2 or n % 2 == 0:
      return False
  else:
      x = math.floor(math.sqrt(n)) + 1
      i = 3
      while i < x:
          if n % i == 0:
              return False
          i += 2
      return True
```

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

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

```
from dataclasses import dataclass
from typing import Optional

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

Schreiben Sie eine Funktion `find_substrings`, die einen Baum `node` und einen String `substr` als Argument nimmt und alle Markierungen, die `substr` als Teilstring enthalten, in einer Liste zurückgibt.
Der Baum soll dabei in *Pre-Order* Reihenfolge durchlaufen werden.
Beispiel:

```
>>> tree = Node("aab", Node("baa", None, None), Node("aba", None, None))
>>> find_substrings(tree, 'ab')
['aab', 'aba']
```

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

Verwenden Sie in den folgenden Teilaufgaben *keine* von Python bereitgestellten Generator-Funktionen *außer* `range`. Die Funktionen `map`, `filter` und `enumerate` sind also z.B. verboten.
Vermeiden Sie unnötigen Speicherverbrauch: Funktionen die Generatoren als Argument nehmen, dürfen diese nicht unnötigerweise in eine Liste umwandeln.
In den folgenden Teilaufgaben müssen Sie *keine* Typannotationen angeben.

(a) Schreiben Sie eine Generator-Funktion `my_enumerate`, die sich wie das von Python bereitgestellte `enumerate` verhält.

Beispiel:

```
>>> list(my_enumerate("abcd"))
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
```

(b) Schreiben Sie eine Generator-Funktion `prefixes`, die eine Liste von ganzen Zahlen als Argument nimmt und alle Anfangsstücke der Liste generiert.

Beispiel:

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

(c) Schreiben Sie eine Generator-Funktion `alternate`, die zwei Generatoren `xs` und `ys` als Argumente nimmt und abwechselnd Elemente von `xs` und `ys` generiert. Ist einer der Generatoren erschöpft so sollen nur noch Elemente des anderen Generators generiert werden bis auch dieser erschöpft ist.

Beispiel:

```
>>> list(alternate(iter("abcdef"), iter([0, 1, 2])))
['a', 0, 'b', 1, 'c', 2, 'd', 'e', 'f']
```

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) Schreiben Sie eine Funktion `twice`, die eine Funktion `f` als Argument nimmt und eine Funktion zurück gibt die `f` zwei mal hintereinander ausführt. Sie können dabei annehmen, dass `f` ein einziges Argument nimmt welches vom gleichen Typ wie `f`'s Rückgabetyp ist.

Beispiel:

```
>>> twice(lambda x: x*2)(5)
20
```

(b) Schreiben Sie eine Funktion `pythagorean_triples`, die eine ganze Zahl `n` als Argument nimmt und eine Liste aller Tripel (a, b, c) aus positiven ganzen Zahlen kleiner n zurückgibt, welche die Bedingung $a^2 + b^2 = c^2$ erfüllen. Die Liste soll dabei keine Duplikate enthalten in folgendem Sinne: wenn (a, b, c) enthalten ist, dann ist (b, a, c) nicht enthalten.

Verwenden Sie hierfür genau eine List-Comprehension.

Beispiel:

```
>>> list(pythagorean_triples(15))
[ (3, 4, 5), (6, 8, 10), (5, 12, 13) ]
```

Sie müssen dabei nicht zwingend die gleiche Reihenfolge und Auswahl an Duplikaten treffen wie im Beispiel zu sehen ist.