Aufgabenblatt 3

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihre Python-Datei bis spätestens **Donnerstag, 03.04. 16:00 Uhr** in TUWEL hoch.
- Beachten Sie bitte folgende Punkte
  - Die Programme müssen syntaktisch korrekt sein. Achten Sie außerdem darauf, dass die Beispiele aus der Angabe zu keinen Abstürzen führen.
  - Wenn Fehler auftreten ziehen wir abhängig von der Schwere der Fehler Punkte ab. Bei entsprechend kleinen Fehlern können auch gar keine Punkte abgezogen werden. Geben Sie daher die bestmögliche eigene Lösung ab, auch wenn diese nicht vollständig richtig ist.
  - Ihre Programme sollen demonstrieren, dass Sie die Themen des Aufgabenblattes beherrschen. Verwenden sie daher keine "Abkürzungen" und beschränken Sie sich auf die Konstrukte die in der Vorlesung vorgestellt wurden.
  - **NEU:** Im Rahmen dieses Übungsblattes sind mehrere Funktionen zu implementieren. Diese können Sie wie gewohnt in die Abgabedatei schreiben. Des weiteren sollen diese Funktionen aufgerufen werden und Ausgaben erstellt werden. Achten Sie bitte darauf, dass diese Aufrufe und Ausgaben nur innerhalb der vorgegebenen Funktionen `exercise1`, `exercise2` und `exercise3` stattfinden! Sie finden entsprechende Kommentare in der Angabedatei.
  - Das Aufgabenblatt enthält 4 Aufgaben, auf welche Sie insgesamt 4 Punkte erhalten können.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Funktionen
- Rekursion

Aufgabe 1 (Funktionen) [1 Punkt]

Ziel dieser Aufgabe ist es, das Erstellen und Verwenden eigener Funktionen zu verinnerlichen, insbesondere die Parameterübergabe und die Rückgabe von Werten.

- Implementieren Sie die beschriebenen Funktionen. Achten Sie darauf, nur in den explizit beschriebenen Funktionen Ausgaben zu erzeugen. Sie können davon ausgehen, dass den Funktionen nur Argumente des jeweils beschriebenen Typs übergeben werden, und müssen dies nicht überprüfen.
- Testen Sie Ihre Implementierung zumindest mit den angegebenen Aufrufen. Geben Sie diese Aufrufe in der Funktion `exercise1()` an. Stellen Sie dabei sicher, dass das Ergebnis der Funktion immer ausgegeben wird – auch wenn in der Funktion selber keine Ausgabe erfolgt.

1. Eine Funktion `count_unfollowed(line, symbol, blocks)`, welche im String `line` alle Vorkommen des Zeichens `symbol` zählt, die nicht direkt vor einem Zeichen aus dem String `blocks` stehen. Gibt die gefundene Anzahl zurück.

   Beispielaufrufe und erwartete Rückgaben:

`count_unfollowed('eis essen kennt keine grenzen', 'e', 'in')`	gibt `2` zurück
`count_unfollowed('2025-02-29', '2', '-500')`	gibt `1` zurück
`count_unfollowed('99 Luftballons, nicht 9', '9', '')`	gibt `3` zurück
`count_unfollowed('Der Sommer kann kommen', 'm', 'mE')`	gibt `2` zurück

2. Eine Funktion `fun_with_numbers(line)`, welche jede zweite Ziffer[1] in `line`, die nicht direkt vor einem Fragezeichen `'?'` steht, aufsummiert und die Summe zurückgibt. Ziffern direkt vor einem `'?'` werden dabei komplett ignoriert und haben auch keinen Einfluss auf die Auswahl "jeder zweiten Ziffer". So würde bei `'12?34/5?678'` die Berechnung lauten `3 + 6 + 8`.

   Beispielaufrufe und erwartete Rückgaben:

`fun_with_numbers('12?34/5?678')`	gibt `17` zurück
`fun_with_numbers('2025?')`	gibt `0` zurück
`fun_with_numbers('1.?2.3.4.5??6.?')`	gibt `6` zurück
`fun_with_numbers('nur 1 oder 2?')`	gibt `0` zurück

3. Eine Funktion `visualize_standings(p1, p2, ratio, length, p1_symb, p2_symb)` welche in einer Zeile folgende Ausgabe erzeugt: Am Beginn der Zeile steht der in `p1` übergebene String in eckigen Klammern, am Ende der Zeile der in `p2` übergebene String, ebenfalls in eckigen Klammern. Zum Beispiel wird aus `'hallo'` dann `'[hallo]'`.

   Dazwischen befindet sich ein aus `length` vielen Zeichen bestehender Balken, der das Verhältnis `ratio` zwischen `p1` und `p2` ausdrückt. Dazu wird zuerst das in `p1_symb` übergebene Zeichen `ratio*length` (abgerundet) oft ausgegeben. Die auf `length` fehlenden Zeichen werden mit dem Zeichen in `p2_symb` aufgefüllt.

   Dabei sollen die folgenden Default-Parameter verwendet werden: `length` ist `10`, `p1_symb` ist `'#'` und `p2_symb` ist `'-'`.

[1] Das gewünschte Muster ist: 1. Ziffer auslassen, 2. Ziffer wählen, 3. Ziffer auslassen, 4. Ziffer wählen, usw.

Beispielaufrufe und erwartete Ausgaben:

`visualize_standings('Player1', 'Player2', 0.5, 10, '#', '-')`
erzeugt die Ausgabe: `[Player1]#####-----[Player2]`

`visualize_standings('Player1', 'Player2', 0.5)`
erzeugt die Ausgabe: `[Player1]#####-----[Player2]`

`visualize_standings('Player1', 'Player2', 0.5, 8)`
erzeugt die Ausgabe: `[Player1]####----[Player2]`

`visualize_standings('Red', 'Blue', 0.75, p1_symb='>', p2_symb=':')`
erzeugt die Ausgabe: `[Red]>>>>>>>:::[Blue]`

`visualize_standings('Ryu', 'Ken', 0.2, 11)`
erzeugt die Ausgabe: `[Ryu]##---------[Ken]`

`visualize_standings('Ryu', 'Ken', 0.09, 10)`
erzeugt die Ausgabe: `[Ryu]----------[Ken]`

Aufgabe 2 (Funktionen kombinieren) [1 Punkt]

Ziel dieser Aufgabe ist es weiterhin, den Umgang mit Funktionen zu verinnerlichen. Implementieren Sie dazu die folgenden vier Funktionen, und testen Sie jeweils Ihre Implementierung. Verwenden Sie zum Testen die Funktion `exercise2()`, und gehen Sie dabei vor wie in Aufgabe 1.
Zu jeder Funktion sind neben der Beschreibung ihres Verhaltens auch noch Annahmen über die Parameterwerte gegeben. Sie können davon ausgehen, dass diese Annahmen stets eingehalten werden. D.h. Sie müssen deren Gültigkeit weder überprüfen, noch muss Ihre Funktion für Argumentwerte, welche diese Annahmen nicht erfüllen, funktionieren.
Für das Beispiel gehen wir zurück zu den Stockwerknummerierungen aus Aufgabenblatt 1, gehen diesmal jedoch davon aus, dass die Stockwerke 4F und 4B vorkommen dürfen.

1. Eine Funktion `validate_floor_name(name)` welche zurückgibt, ob es sich bei dem String `name` um eine gültige Stockwerksbezeichnung handelt. Das heißt: Das letzte Zeichen des Strings ist entweder ein `'B'` oder ein `'F'`, und alle anderen Zeichen sind Ziffern, wobei das erste Zeichen keine `'0'` ist.

   Beispielaufrufe und erwartete Rückgaben:

   `validate_floor_name('1F')` gibt `True` zurück

   `validate_floor_name('0F')` gibt `False` zurück

   `validate_floor_name('F')` gibt `False` zurück

   `validate_floor_name('2A')` gibt `False` zurück

   `validate_floor_name('5B')` gibt `True` zurück

   `validate_floor_name('12F')` gibt `True` zurück

   `validate_floor_name('0123F')` gibt `False` zurück

2. Eine Funktion `translate_floor_plan(name)`, welche eine gültige japanische Stockwerksbezeichnung in eine Zahl übersetzt. Wir gehen dabei etwas einfacher vor als in Aufgabenblatt 1: Endet der Name auf `'F'` wird die davor stehende Zahl um den Wert 1 verkleinert zurückgegeben, endet der Name auf `'B'` wird der negierte Wert der davor stehenden Zahl zurückgegeben. Die Rückgabe muss vom Typ `str` (nicht `int`) sein. Sie dürfen annehmen, dass `name` eine gültige Stockwerksbezeichnung ist und brauchen dies nicht überprüfen.

   Beispielaufrufe und erwartete Rückgaben:

   `translate_floor_plan('1F')` gibt `'0'` zurück

   `translate_floor_plan('1B')` gibt `'-1'` zurück

   `translate_floor_plan('10F')` gibt `'9'` zurück

3. Eine Funktion `print_translation(*floors)`, welche eine unbestimmte Anzahl an Strings erhält, jeden dieser Strings **mit Hilfe der Funktion** `validate_floor_name` daraufhin überprüft, ob es sich um eine gültige japanische Stockwerksbezeichnung handelt, und dann folgende Ausgaben erzeugt:

   Für das erste gültige Stockwerk wird die Zeile `'Da lief ich in den x. Stock'` ausgegeben, wobei `x` die **mittels** `translate_floor_plan` umgerechnete Stockwerksnummer ist.

   Für alle weiteren gültigen Stockwerke wird jeweils eine Zeile

   `'dann lief ich in den x. Stock'` ausgegeben.

   Am Ende wird entweder die Zeile `'... und aus.'` ausgegeben, wenn mindestens eine gültige Stockwerksbezeichnung gefunden wurde, und sonst die Zeile `'Kein Stock'`.

   Verwenden Sie die Funktionen `validate_floor_name` und `translate_floor_plan`, um diese Aufgabe zu lösen.

   Beispielaufrufe und erwartete Ausgaben:

   `print_translation('0F', '1F', '2F', '3F', '4', '5F', '06F')` erzeugt die Ausgabe:

```
Da lief ich in den 0. Stock
dann lief ich in den 1. Stock
dann lief ich in den 2. Stock
dann lief ich in den 4. Stock
... und aus.
```

   `print_translation('1B', 'Dach', '7F', '5B', '12F', '00')` erzeugt die Ausgabe:

```
Da lief ich in den -1. Stock
dann lief ich in den 6. Stock
dann lief ich in den -5. Stock
dann lief ich in den 11. Stock
... und aus.
```

   `print_translation('0F', '0B', '1', '-2F')` erzeugt die Ausgabe:

```
Kein Stock
```

   `print_translation()` erzeugt die Ausgabe:

```
Kein Stock
```

Aufgabe 3 (Code mittels Funktionen (und Schleifen) strukturieren) [1 Punkt]

Ziel dieser Aufgabe ist es einerseits, gegebenen Code zu verstehen, und gleichzeitig den Einsatz von (Schleifen und) Funktionen zur Strukturierung sowie leichteren Les- und Wartbarkeit des Codes zu üben.
Dazu ist eine schlecht programmierte Funktion `word_guess_spaghetti(secret)` gegeben, welche ein Spiel zum Erraten des in `secret` übergebenen Wortes implementiert (zu einer genauen Beschreibung des Spielablaufs führen Sie die Funktion aus und/oder lesen Sie den Programmcode).
Warum ist die Implementierung dieser Funktion schlecht? Auch wenn der Spielablauf relativ geradlinig durch die Funktion abgebildet ist, so enthält die Funktion viel redundanten Code (die selbe Funktionalität wird an verschiedenen Stellen implementiert), und ist in verschiedenen Aspekten nur schwer erweiterbar (z.B. ist die Anzahl der Runden auf sechs beschränkt - eine Anpassung der Rundenanzahl wäre ziemlich aufwendig).
Überlegen Sie sich, wie sich das Spiel durch den Einsatz von Schleifen und Funktionen besser implementieren lässt, und geben Sie eine entsprechende Implementierung an. Achten Sie bei Ihrer Implementierung insbesondere darauf, redundanten Code zu vermeiden.

- Implementieren Sie dazu die Funktion `better_word_guess(secret, max_guesses)`.

  Das Verhalten der Funktion bei Aufruf von `better_word_guess(secret, 6)` soll ident sein zu dem Verhalten von `word_guess_spaghetti(secret)`.

- Erstellen Sie dabei so viele Funktionen, wie Sie für sinnvoll erachten. Sie können die Parameter sowie die Rückgabewerte Ihrer Funktionen frei wählen.

- Die Verwendung globaler Variablen und ähnlicher Konstrukte ist verboten. Funktionen dürfen Informationen ausschließlich über Parameter und Rückgabewerte austauschen.

- Ihre neuen Funktionen werden im Rahmen des Spiels aufgerufen. Das garantiert, dass die übergebenen Parameter gewisse Werte besitzen/Bedingungen einhalten (z.B. kann es sein, dass eine übergebene Zahl immer > 0 ist, oder dass ein übergebener String immer eine bestimmte (minimale) Länge hat). Es ist absolut in Ordnung, wenn Ihre Funktionen nur in diesen bestimmten Fällen fehlerfrei funktionieren.

  Überlegen Sie sich jedoch welche Annahmen es sind, unter denen die Funktion implementiert wurde, und welche Bedingungen die Parameter erfüllen müssen. Dokumentieren Sie diese Bedingungen zu jeder Funktion (eine Vorlage finden Sie in der Angabedatei `blatt3.py`).

- *[freiwillig]* Die gegebene Version des Spiels ist sehr eingeschränkt: Wörter mit mehr als sechs verschiedenen Zeichen können nie erraten werden. Ändern Sie das Spiel so ab, dass es für beliebige Wörter sinnvoll gespielt werden kann.

  Ein Weg dies zu erreichen ist, die Implementierung so zu ändern, dass die Anzahl an gespielten Runden durch einen Parameter gesteuert werden kann. Dadurch kann die Anzahl der Runden an die Anzahl der Zeichen im Wort angepasst werden. Normalerweise wird das Spiel aber so gespielt, dass Runden nur dann gezählt werden, wenn ein Zeichen geraten wurde, das nicht Teil des Wortes ist (nur falsches Raten verbraucht ein "Leben"). Diese Regel zu implementieren stellt eine zweite Möglichkeit dar, die Aufgabe zu lösen.

  Wenn die Implementierung des Spiels ausreichend verbessert wurde, sollten sich beide Änderungen relativ einfach umsetzen lassen – auch beide gemeinsam, was sicherlich die schönste Lösung darstellt.

Aufgabe 4 (Rekursion) [1 Punkt]

In der Angabedatei (`blatt3.py`) finden Sie die Implementierung dreier rekursiver Funktionen. Vor jeder Funktion finden Sie einen kurzen Kommentarabschnitt der Form

```
# Funktionalität:
# Basisfall:
# "Verkleinerung" des Problems im Rekursionsschritt:
# Berechnung der Lösung im Rekursionsschritt:
```

Verwenden Sie diese Zeilen, um die folgenden Eigenschaften der gegebenen Funktionen in jeweils **1-2 kurzen Sätzen** zu beschreiben:

- Welche Funktionalität bietet die Funktion (was wird berechnet/zurückgegeben/ausgegeben)?
- Den Basisfall bzw. die Basisfälle, falls es mehr als einen geben sollte. (Fügen Sie in dem Fall bitte für jeden weiteren Basisfall eine neue Kommentarzeile ein.)
- Wie trägt der/wie tragen die Rekursionsschritte zum Erreichen der Abbruchbedingung/des Basisfalls bei?
- Wie wird in den Rekursionsschritten die Lösung des aktuellen Problems aus den Lösungen der rekursiven Aufrufe berechnet?

Sie können davon ausgehen, dass die in der Angabedatei beschriebenen Annahmen für die Argumente eingehalten werden. Ein Beispiel welches die Aufgabenstellung demonstrieren soll ist bereits in der Angabedatei vorgegeben.
Sie können zur Beantwortung der Fragen die Funktionen natürlich ausführen. Achten Sie jedoch bitte darauf, dass in Ihrer Abgabedatei keine Aufrufe dieser Funktionen mehr enthalten sind.