Aufgabe 5 - Unit Tests mit Exceptions

Ziel

- Sie können Testklassen auf der Basis von `pytest` erstellen und anwenden.

Hinweis

- Unit-Tests werden häufig mit agilen Programmierverfahren verwendet, speziell der testgetrieben Programmierung TDD.
  - Dabei werden zuerst die Testfälle (Black-Box) festgelegt und dann der dazu nötige Testcode erstellt.
  - Erst dann wird die eigentliche Klasse erstellt und laufende gegen die Testklasse geprüft.
  - Der Code wird solange angepasst, bis alle Tests fehlerfrei absolviert werden (s.a. Theorie, Kapitel **Unit Testing**).
- Wir verwenden hier den Unit-Test in Bezug auf schon bestehende Klassen.

Auftrag

1. Vorbereiten der Testdatei `grade_list_test.py`

- Erstellen Sie – wenn nicht schon erfolgt – ein Projekt für `pytest` in PyCharm.
- Fügen Sie dem Unit Test eine Funktion `testee()` hinzu, welche eine `GradeList` zurück gibt.
- Versehen Sie die Funktion mit der Annotation `@pytest.fixture`.

2. Testen der Methode `get_max_grade_count()`

- Wir prüfen, ob der gelieferte Wert der Grösse der Liste entspricht.
- Fügen Sie die Methode `test_max_grade_count()` hinzu.
- Die Methode verwendet das `pytest`-Schlüsselwort `assert` zum Vergleichen des gewünschten mit dem aktuellen Wert.
- Prüfen Sie mit `assert`, ob das Ergebnis vom Aufruf `testee.get_max_groesse()` mit dem konstanten Wert 3 übereinstimmt.
- Führen Sie den Unit Test aus. Das Ergebnis des Tests wird im linken Teil des Programmfensters angezeigt.
- Wenn Sie alles richtig gemacht haben, finden Sie einen orangen Balken und im unteren Bereich des Fensters einen Hinweis auf den Fehler.
- Ersetzen Sie nun die Zahl durch den konstanten Wert 5.
- Speichern Sie die Klasse und führen Sie den Test erneut aus.
- Nun muss als Ergebnis ein grüner Balken angezeigt werden.

3. Test der Methode `get_current_grade_count()`

- Wir prüfen, ob die Methode sich in verschiedenen Situationen korrekt verhält.
- Zu Beginn muss die Grösse 0 sein, nach dem Zufügen eines Elements muss der Wert 1 sein und nach zufügen von mehr als 5 Elemente muss der Wert 5 sein.
- Es sind hier 3 Testmethoden nötig, um alle diese Fälle sicherzustellen.
- Erstellen Sie die 3 Methoden `test_list_is_empty()`, `test_list_is_not_empty()` und

`test_list_is_full()`.
- Beachten Sie, dass die Methode `add_grade()` eine Exception wirft, die Sie hier fangen aber nicht behandeln müssen.
    - Dies können Sie entweder mit `try..except` oder besser mit einem Abschnitt `with pytest.raises(…Error):` erreichen.
    - Lesen Sie kurz nach → Assertions about expected exceptions
- Führen Sie nun den Test durch und prüfen Sie das Ergebnis.

4. Testen der Methode `add_grade()`

- Wir prüfen eine Methode, die im Fehlerfall eine Exception wirft. Pytest kann auch dieses Verhalten prüfen.
- Erstellen Sie 2 Methoden `test_add_valid()` und `test_add_invalid()`
- Bei `test_add_valid()` sollten Sie wiederum das `with` Statement verwenden.
- Führen Sie nun den Test durch und prüfen Sie das Ergebnis.

5. Test der Methode `get_grade()`

- Codieren Sie für die folgenden Testfälle Ihre eigenen Test-Methoden.

#	Eingabewert	Testwert	Erwartetes Ergebnis
1	-1	-	IndexError wird geworfen
2	0	5.0f	5.0f
3	5	-	IndexError wird geworfen

Abgabe

Geben Sie Ihren Code via Moodle ab.