LU07.L07 - Sicherheitsnetz zuerst

Schritt 1: Verhalten erkunden

```
notenschnitt([18, 20, 16], 20)  ->  5.5
notenschnitt([0, 0], 20)        ->  1.0
notenschnitt([20, 20], 20)      ->  6.0
notenschnitt([25], 20)          ->  6.0     (Deckelung greift)
notenschnitt([10], 0)           ->  1       (int, nicht float)
notenschnitt([13, 14], 20)      ->  4.5
notenschnitt([], 20)            ->  ZeroDivisionError
```

Schritt 2: Festhalten

```python
import pytest
from noten import notenschnitt

@pytest.mark.parametrize("punkte, maximum, erwartet", [
    ([18, 20, 16], 20, 5.5),
    ([0, 0],       20, 1.0),
    ([20, 20],     20, 6.0),
    ([25],         20, 6.0),
    ([10],          0, 1),
    ([13, 14],     20, 4.5),
])
def test_notenschnitt_bleibt_gleich(punkte, maximum, erwartet):
    assert notenschnitt(punkte, maximum) == erwartet

def test_leere_liste_wirft_bisher():
    with pytest.raises(ZeroDivisionError):
        notenschnitt([], 20)
```

Der letzte Test ist der wichtigste: Auch ein Absturz ist Verhalten und muss festgehalten werden. Wer ihn beim Refactoring versehentlich «wegrepariert», ändert das Verhalten - und merkt es ohne diesen Test nicht.

Schritt 3: Refactoring

```python
MIN_NOTE = 1
MAX_NOTE = 6


def auf_halbe_note(note):
```

```
    return round(note * 2) / 2


def notenschnitt(punkte, maximum):
    if maximum == 0:
        return MIN_NOTE
    erreichter_anteil = sum(punkte) / len(punkte) / maximum
    note = MIN_NOTE + (MAX_NOTE - MIN_NOTE) * erreichter_anteil
    return auf_halbe_note(min(note, MAX_NOTE))
```

Alle sieben Tests bleiben grün. Die Formel `1 + 5 * …` ist jetzt als «lineare Skala von 1 bis 6» erkennbar, und `auf_halbe_note` beantwortet die Frage nach `round(note * 2) / 2`.

Schritt 4: Bewertung

Beobachtung	Fachlich problematisch?	Was wäre zu tun
`maximum = 0` liefert `1`	ja - das ist stillschweigend eine Notengebung für eine unmögliche Situation	`ValueError` werfen, damit der Aufrufer den Konfigurationsfehler bemerkt
`[]` wirft `ZeroDivisionError`	ja - die Meldung sagt nichts über den Fall aus	eigene Prüfung mit klarer Fehlermeldung
`1` statt `1.0`	nein, in Python vergleichbar - kann aber in Ausgabe und JSON auffallen	`float(MIN_NOTE)` zurückgeben
Punkte über dem Maximum werden auf 6 gedeckelt	je nach Reglement gewollt (Bonuspunkte)	mit der Fachseite klären

Warum zuerst festhalten und nicht gleich reparieren? Weil sonst zwei Änderungen gleichzeitig passieren. Wenn danach etwas nicht mehr stimmt, wissen Sie nicht, ob das Refactoring oder der Bugfix schuld ist. Die Reihenfolge lautet immer: festhalten, refactoren (Tests grün), dann in einem eigenen Commit das Verhalten bewusst ändern und den Test anpassen.

M323-LU07, M323-D1I, M323-D1A