LU03f - Parametrisierte Tests

Auf einen Blick
Oft unterscheiden sich Tests nur in den Testdaten. Statt die gleiche Testfunktion mehrmals zu kopieren, schreiben Sie sie einmal und geben die Testfälle als Tabelle an. PyTest führt die Funktion für jede Zeile der Tabelle als eigenen Test aus.

Lernziele

Sie können

- mehrere Testfälle mit `@pytest.mark.parametrize` in einer Testfunktion zusammenfassen,
- parametrisierte Tests für gültige und ungültige Eingaben schreiben,
- den Testfällen lesbare Namen geben.

Das Problem: kopierte Tests

```
def test_minmax_normal():
    assert minmax([3, 1, 2]) == (1, 3)


def test_minmax_one_element():
    assert minmax([5]) == (5, 5)


def test_minmax_negative():
    assert minmax([-4, -9, -1]) == (-9, -1)
```

Die drei Tests sind identisch bis auf die Daten. Kommt ein Testfall dazu, wird nochmals kopiert. Das verstösst gegen das Prinzip DRY (Don't Repeat Yourself).

Die Lösung: parametrize

```
import pytest
from minmax import minmax


@pytest.mark.parametrize('numbers, expected', [
    ([3, 1, 2], (1, 3)),
    ([5], (5, 5)),
    ([-4, -9, -1], (-9, -1)),
    ([2, 2, 2], (2, 2)),
```

```
    ([1.5, 0.5], (0.5, 1.5)),
])
def test_minmax(numbers, expected):
    assert minmax(numbers) == expected
```

- Der erste Parameter ist ein Text mit den **Namen** der Testdaten, durch Komma getrennt.
- Der zweite Parameter ist eine **Liste von Tupeln**. Jedes Tupel ist ein Testfall.
- Die Testfunktion erhält die Werte über gleichnamige Parameter.

PyTest zeigt jeden Testfall einzeln an:

```
test_minmax.py::test_minmax[numbers0-expected0] PASSED
test_minmax.py::test_minmax[numbers1-expected1] PASSED
test_minmax.py::test_minmax[numbers2-expected2] PASSED
...
```

Lesbare Namen mit pytest.param

Die automatischen Namen wie `numbers0-expected0` sagen wenig. Mit `pytest.param` und `id` geben Sie jedem Testfall einen Namen:

```
@pytest.mark.parametrize('numbers, expected', [
    pytest.param([3, 1, 2], (1, 3), id='normal'),
    pytest.param([5], (5, 5), id='ein Element'),
    pytest.param([-4, -9, -1], (-9, -1), id='nur negative'),
])
def test_minmax(numbers, expected):
    assert minmax(numbers) == expected
```

```
test_minmax.py::test_minmax[normal] PASSED
test_minmax.py::test_minmax[ein Element] PASSED
test_minmax.py::test_minmax[nur negative] PASSED
```

Ungültige Eingaben parametrisieren

Auch Tests mit `pytest.raises` (LU03e) lassen sich parametrisieren:

```
@pytest.mark.parametrize('amount', [0, -0.01, -100])
def test_deposit_invalid_amount(amount):
    account = BankAccount('Anna', 100.0)
    with pytest.raises(ValueError):
        account.deposit(amount)
```

Trennen Sie gültige und ungültige Testfälle in zwei Testfunktionen. Sonst braucht der Test ein `if`, und ein Test mit Verzweigungen ist selbst fehleranfällig.

Verbindung zum Testfall-Dokument

Die Tabelle in `parametrize` entspricht den Zeilen einer Testfallbeschreibung aus LU01c: Eingabe und SOLL. So lässt sich eine Liste von Testfällen direkt in einen automatisierten Test übertragen. Dieses Vorgehen heisst **datengetriebenes Testen** (siehe LU02a).

Kontrollfragen

1. Wie viele Tests meldet PyTest für eine Testfunktion mit fünf Tupeln in `parametrize`?
2. Schreiben Sie `@pytest.mark.parametrize` für `factorial` mit den Testfällen 0, 1 und 7.
3. Warum sollten gültige und ungültige Testfälle nicht in derselben parametrisierten Funktion stehen?

M450-LU03, M450-C1A

Marcel Suter

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m450/learningunits/lu03/parametrize

Last update: **2026/11/17 07:58**