LU04a - Fixture

Auf einen Blick
Eine **Fixture** (deutsch: Vorrichtung, Halterung) bereitet alles vor, was ein Test braucht, und räumt danach wieder auf. In PyTest ist eine Fixture eine Funktion mit dem Dekorator `@pytest.fixture`. Ein Test fordert sie an, indem er ihren Namen als Parameter angibt. So haben alle Tests die gleiche, wiederholbare Ausgangslage, ohne dass der Vorbereitungscode kopiert werden muss.

Lernziele

Sie können

- eigene Fixtures definieren und in Tests verwenden,
- Fixtures kombinieren und mit `yield` nach dem Test aufräumen,
- gemeinsame Fixtures in `conftest.py` ablegen,
- die eingebauten Fixtures `capsys`, `monkeypatch` und `tmp_path` gezielt einsetzen.

Wozu Fixtures?

Ein Test läuft in vier Schritten ab:

Schritt	Was passiert	In PyTest
Vorbereiten	Objekte erzeugen, Dateien oder Datenbanken mit Testdaten füllen	Fixture (vor `yield` oder `return`)
Ausführen	Die zu testende Funktion aufrufen	Testfunktion (Act)
Auswerten	IST mit SOLL vergleichen	Testfunktion (Assert)
Aufräumen	Alle Änderungen rückgängig machen, damit andere Tests nicht beeinflusst werden	Fixture (nach `yield`)

Schreiben Sie die Vorbereitung in jede Testfunktion, wiederholt sich derselbe Code dutzende Male. Ändert sich der Konstruktor einer Klasse, müssen Sie alle Tests anpassen. Mit einer Fixture ändern Sie nur eine Stelle.

Eigene Fixtures definieren

```
import pytest
from library import Customer, Library


@pytest.fixture
def customer_max():
    return Customer('Max')
```

```python
@pytest.fixture
def customer_moritz():
    return Customer('Moritz')


@pytest.fixture
def library():
    return Library()


def test_add_customers(library, customer_max, customer_moritz):
    library.add_customer(customer_max)
    library.add_customer(customer_moritz)
    assert len(library.customers) == 2
```

- PyTest sieht die Parameter `library`, `customer_max` und `customer_moritz`, ruft die gleichnamigen Fixtures auf und übergibt deren Rückgabewerte.
- Jeder Test erhält **neue** Objekte. Ändert ein Test die Bibliothek, merkt der nächste Test nichts davon.

Fixtures und Testfunktionen, die **nicht** in einer Klasse stehen, haben **keinen** Parameter `self`. Sonst meldet PyTest den Fehler `fixture 'self' not found`.

Fixtures kombinieren

Eine Fixture kann selbst andere Fixtures anfordern. So bauen Sie komplexere Ausgangslagen schrittweise auf:

```python
@pytest.fixture
def library_with_customers(library, customer_max, customer_moritz):
    library.add_customer(customer_max)
    library.add_customer(customer_moritz)
    return library


def test_find_customer(library_with_customers):
    customer = library_with_customers.find_customer('Moritz')
    assert customer.name == 'Moritz'


def test_find_unknown_customer(library_with_customers):
    assert library_with_customers.find_customer('Lempel') is None
```

Aufräumen mit yield

Muss nach dem Test etwas rückgängig gemacht werden, verwenden Sie `yield` statt `return`. Der Code nach `yield` läuft nach dem Test, auch wenn der Test fehlgeschlagen ist.

```
@pytest.fixture
def app():
    shutil.copy('./files/books.bak.json', './files/books.json')   #
Vorbereiten: Testdaten herstellen
    app = create_app()
    app.testing = True
    yield app                                                     # Hier
läuft der Test
    shutil.copy('./files/books.bak.json', './files/books.json')   #
Aufräumen: Originaldaten zurück


@pytest.fixture
def client(app):
    return app.test_client()
```

Dieses Beispiel stammt aus einer Flask-Applikation: Vor jedem Test wird die Datei `books.json` aus einer Sicherung wiederhergestellt, damit jeder Test mit denselben Büchern beginnt.

Gemeinsame Fixtures in conftest.py

Brauchen mehrere Testdateien dieselbe Fixture, legen Sie sie in die Datei `conftest.py` im Projektordner. PyTest lädt diese Datei automatisch. Ein `import` ist nicht nötig.

```
# conftest.py
import pytest
from library import Library


@pytest.fixture
def library():
    return Library()
```

Testdaten aus einer Datei lesen

Eine Fixture kann Testdaten aus einer Datei laden. Fehlt die Datei, soll der Test mit einer verständlichen Meldung abbrechen:

```
import json
import pytest
```

```python
@pytest.fixture
def test_values():
    try:
        with open('./testdata.json', encoding='utf-8') as json_file:
            return json.load(json_file)
    except FileNotFoundError:
        pytest.fail('Die Datei "testdata.json" fehlt. Kontaktieren Sie Ihre Lehrperson.')
```

Mehr zu Testdaten erfahren Sie in LU12.

Eingebaute Fixtures

PyTest bringt viele Fixtures mit. Diese kennen Sie bereits aus LU03c:

Fixture	Zweck
`capsys`	Ausgaben im Terminal (`print`) abfangen und prüfen
`monkeypatch`	Funktionen, Attribute, Dictionary-Einträge und Umgebungsvariablen während des Tests ersetzen (LU04b)
`tmp_path`	Einen leeren, temporären Ordner für Dateien bereitstellen

Eine vollständige Liste zeigt der Befehl `pytest –fixtures`.

Kontrollfragen

1. Was ist der Unterschied zwischen `return` und `yield` in einer Fixture?
2. Fünf Testdateien brauchen dasselbe `Library`-Objekt. Wo definieren Sie die Fixture?
3. Ein Test meldet `fixture 'self' not found`. Was ist falsch?
4. Warum erzeugt eine Fixture für jeden Test ein neues Objekt, statt ein Objekt für alle Tests zu teilen?

Weiterführende Links

- PyTest: How to use fixtures

M450-LU04, M450-C1A

Marcel Suter

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

Permanent link:
https://wiki.bzz.ch/modul/m450/learningunits/lu04/fixture

Last update: **2026/11/17 08:02**