LU04a - Fixture

Eine *Fixture* (deutsch: Vorrichtung, Halterung) dienen zur Initialisierung von Tests. Sie bieten eine statische Basis, damit Test zuverlässig ausgeführt werden und konsistente, wiederholbare Resultate erzeugen.

Was sind Fixtures?

Bei der Durchführung von Tests müssen wir sicherstellen, dass diese immer wieder die gleichen Resultate erzeugen. Dazu müssen die Tests immer in der gleichen Umgebung mit den gleichen Daten durchgeführt werden. Fixtures helfen uns diese gleich bleibende Basis für unsere Testfälle sicherzustellen.

Die Durchführung eines Tests kann in vier Schritte unterteilt werden:

1. Vorbereitung: Wir bereiten die Umgebung für die Durchführung des Tests vor. Dazu gehören:
    - Objekte erzeugen
    - Daten in Datenbanken oder Dateien bereitstellen
    - Umgebungsvariablen wie URL, Pfade, ... definieren
2. Ausführen: Wir führen die zu testende Funktion aus und nehmen die Resultate entgegen.
3. Auswerten: Wir vergleichen die tatsächlichen Resultate mit den erwarteten Resultaten. Anhand dieses Vergleichs können wir bewerten, ob der Test erfolgreich war oder nicht.
4. Aufräumen: Zuletzt sollte der Test alle Veränderungen am System rückgängig machen. Dadurch verhindern wir, dass die Resultate anderer Tests beeinflusst werden.

Im engeren Sinne des Wortes sind Fixtures Vorbereitungsschritte für die Durchführung eines Tests. Wir können aber Fixtures auch etwas weiter fassen. Es handelt sich um alle nötigen Schritte, rund um den eigentlichen Test. Also sowohl Vorbereitung als auch Ausführung und Aufräumarbeiten.

Fixtures in PyTest

Siehe auch pytest fixtures

PyTest bietet uns ein umfangreiches System zum Einsatz solcher Fixtures an, die wir in den verschiedenen Schritten einsetzen können.

capsys und monkeypatch

In der letzten Learning Unit haben wir bereits mit `capsys` und `monkeypatch` gearbeitet, um die Ein- und Ausgaben zu kontrollieren. Mittels `capsys` haben wir die Ausgaben einer Funktion gelesen. Zum Bereitstellen von Benutzereingaben, haben wir `monkeypatch` genutzt.

Monkeypatch kann mehr als nur Benutzereingaben bereitstellen. Mit Hilfe dieser Fixture können wir

- Attribute von Objekten verändern
- Elemente von Dictionaries ändern und löschen
- Umgebungsvariablen setzen und löschen

Dadurch können wir zum Beispiel Situationen erzeugen, die beim normalen Ablauf des Programms nicht auftreten können oder sollten.

Eigene Fixtures definieren

Neben den von PyTest vorgegebenen Fixtures, können wir auch eigene Funktionen schreiben und als Fixtures nutzen. Ein typischer Einsatz solcher Funktionen ist das Bereitstellen von Objekten und Daten für die Tests. Dadurch stellen wir beliebig vielen Testfunktionen eine einheitliche Datenbasis bereit.

Beispiele

Objekte bereitstellen

Für die Tests in einer Applikation sollen zwei Objekte der Klasse `Customer` bereitgestellt werden.

```
@pytest.fixture
def customer_max(self):
    return Customer('Max', None)

@pytest.fixture
def customer_moritz(self):
    return Customer('Moritz', None)

def test_add_customers(self, customer_max, customer_moritz):
    library = Library()
    library.add_customer(customer_max)
    library.add_customer(customer_moritz)
    assert len(library.customers) == 2   # Checks the number of entries in
the list
```

In der Testfunktion `test_add_customers` werden die beiden Fixtures als Parameter angegeben. Pytest stellt selber sicher, dass die entsprechenden Fixtures ausgeführt werden.

Testdaten aus einer Datei lesen

```
def test_add_entries1(monkeypatch, capsys, version, test_values):
    ....


@pytest.fixture
```

```
def test_values():
    try:
        with open('./testdata.json') as json_file:
            data = json.load(json_file)
            return data
    except FileNotFoundError as ex:
print('\n=================================================================
=======')
        print('WICHTIG: Keine Testdaten gefunden. Kontaktieren Sie SOFORT
Ihre Lehrperson')
print('=================================================================
=====\n\n')
        raise FileNotFoundError


@pytest.fixture
def version():
    collection = echarge.init_accounts()
    if collection is None:
        return 'UNBEKANNT'
    if isinstance(collection, list):
        return 'BASIS'
    if isinstance(collection, dict):
        return 'ERWEITERT'
    return 'UNBEKANNT'
```

Flask-APP als Fixture

```
@pytest.fixture()
def app():
    app = create_app()
    shutil.copy('./files/books.bak.json', './files/books.json')
    yield app


@pytest.fixture()
def client(app):
    app.testing = True
    return app.test_client()


@pytest.fixture()
def runner(app):
    return app.test_cli_runner()
```

M450-LU04