LU16.A01 – Behave Grundlagen

Erstelle mit **Behave** zwei kleine BDD-Übungen inkl. Feature-Dateien und Schrittdefinitionen. Arbeite sauber in der üblichen Projektstruktur (`features/` und `features/steps/`) und nutze die Vorlagen unten.

Projektstruktur (für beide Aufträge)

```
behave_project/
 ├── app/                 <-- (Produktivcode hier)
 │    ├── converter.py
 │    └── auth.py
 └── features/
      ├── temperature.feature
      ├── login.feature
      └── steps/
           ├── temperature_steps.py
           └── login_steps.py
```

Auftrag 1 – Temperatur-Umrechnung (Celsius ↔ Fahrenheit)

Ziel: Schreibe BDD-Tests für eine einfache Umrechnungsfunktion. Teste korrekte Werte und Fehlerverhalten (z. B. ungültige Eingaben).

Feature-Datei

Datei: `features/temperature.feature`

```
Feature: Temperaturumrechnung
  Damit ich Temperaturwerte standardisiert anzeigen kann
  möchte ich zwischen Celsius und Fahrenheit korrekt umrechnen.

  Scenario: Celsius zu Fahrenheit
    Given eine Celsius-Temperatur von 0
    When ich nach Fahrenheit umrechne
    Then erhalte ich 32.0 Fahrenheit

  Scenario: Fahrenheit zu Celsius
    Given eine Fahrenheit-Temperatur von 212
```

```
    When ich nach Celsius umrechne
    Then erhalte ich 100.0 Celsius

  Scenario: Ungültige Eingabe (String)
    Given eine ungültige Eingabe "heiss"
    When ich eine Umrechnung ausführe
    Then wird ein ValueError ausgelöst
```

Schrittdefinitionen (Vorlage)

Datei: `features/steps/temperature_steps.py`

```python
from behave import given, when, then
from app.converter import c_to_f, f_to_c

@given("eine Celsius-Temperatur von {c:d}")
def step_given_celsius(context, c):
    context.input_c = c

@given("eine Fahrenheit-Temperatur von {f:d}")
def step_given_fahrenheit(context, f):
    context.input_f = f

@given('eine ungültige Eingabe "{text}"')
def step_given_invalid(context, text):
    context.invalid = text

@when("ich nach Fahrenheit umrechne")
def step_when_to_f(context):
    context.result = c_to_f(context.input_c)

@when("ich nach Celsius umrechne")
def step_when_to_c(context):
    context.result = f_to_c(context.input_f)

@when("ich eine Umrechnung ausführe")
def step_when_convert_any(context):
    try:
        context.result = c_to_f(context.invalid)
    except Exception as exc:
        context.exc = exc

@then("erhalte ich {expected:float} Fahrenheit")
def step_then_f(context, expected):
    assert abs(context.result - expected) < 1e-6

@then("erhalte ich {expected:float} Celsius")
def step_then_c(context, expected):
```

```
    assert abs(context.result - expected) < 1e-6

@then("wird ein ValueError ausgelöst")
def step_then_error(context):
    assert isinstance(getattr(context, "exc", None), ValueError)
```

Produktivcode (Vorlage)

Datei: `app/converter.py`

```
def c_to_f(celsius):
    if not isinstance(celsius, (int, float)):
        raise ValueError("celsius must be a number")
    return celsius * 9.0 / 5.0 + 32.0

def f_to_c(fahrenheit):
    if not isinstance(fahrenheit, (int, float)):
        raise ValueError("fahrenheit must be a number")
    return (fahrenheit - 32.0) * 5.0 / 9.0
```

Auftrag 2 – Login/Authentifizierung (Happy & Error Paths)

Ziel: Schreibe BDD-Tests für einen sehr einfachen Login-Workflow. Teste: erfolgreicher Login, falsches Passwort, unbekannter Benutzer.

Feature-Datei

Datei: `features/login.feature`

```
Feature: Benutzer-Login
  Um den Zugang zu schützen
  möchte ich, dass sich Benutzende mit Benutzername und Passwort anmelden.

  Background:
    Given es existiert ein Benutzer "alice" mit Passwort "secret"

  Scenario: Erfolgreicher Login
    When ich mich mit Benutzer "alice" und Passwort "secret" anmelde
    Then bin ich eingeloggt

  Scenario: Falsches Passwort
    When ich mich mit Benutzer "alice" und Passwort "wrong" anmelde
    Then ist der Login abgelehnt

  Scenario: Unbekannter Benutzer
```

```
    When ich mich mit Benutzer "bob" und Passwort "irrelevant" anmelde
    Then ist der Login abgelehnt
```

Schrittdefinitionen (Vorlage)

Datei: `features/steps/login_steps.py`

```python
from behave import given, when, then
from app.auth import AuthService

@given('es existiert ein Benutzer "{username}" mit Passwort "{password}"')
def step_given_user_exists(context, username, password):
    context.auth = AuthService()
    context.auth.register(username, password)

@when('ich mich mit Benutzer "{username}" und Passwort "{password}"
anmelde')
def step_when_login(context, username, password):
    context.login_result = context.auth.login(username, password)

@then("bin ich eingeloggt")
def step_then_logged_in(context):
    assert context.login_result is True

@then("ist der Login abgelehnt")
def step_then_denied(context):
    assert context.login_result is False
```

Produktivcode (Vorlage)

Datei: `app/auth.py`

```python
class AuthService:
    def __init__(self):
        self._users = {}

    def register(self, username, password):
        if not username or not password:
            raise ValueError("username and password required")
        self._users[username] = password

    def login(self, username, password):
        if username not in self._users:
            return False
        return self._users[username] == password
```

Ausführung

Im Projekt-Hauptordner:

```
pip install behave
behave
```

Erwartung: Beide Features laufen grün. Erweitere optional weitere Szenarien (z. B. Trimmen von Eingaben, minimale Passwortlänge, Randfälle bei Temperaturen).

MXXX-LU08