

LU08.A01 - Behave Grundlagen



Erstelle mit **Behave** zwei kleine BDD-Übungen inkl. Feature-Dateien und Schrittdefinitionen. **Wichtig:** Die Vorlagen enthalten **nur Gerüste und Hinweise**. Implementiere die Logik **selbst**. Arbeite in der üblichen Struktur (features/ und features/steps/).

Projektstruktur (für beide Aufträge)

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

Hinweise allgemein:

- behave erkennt Szenarien in .feature-Dateien (Gherkin). Kommentare beginnen mit #.
- Schrittdefinitionen befinden sich in features/steps/*.py und nutzen Dekorenoren @given/@when/@then.
- Gemeinsame Daten im Szenario könnt ihr im context-Objekt speichern (z. B. context.result).
- Hardcodierte Zahlen in Szenarien sind ok für den Einstieg (z. B. 0°C → 32°F). Später könnt ihr Scenario Outline nutzen.

Auftrag 1 - Temperatur-Umrechnung (Celsius ↔ Fahrenheit)

Ziel: BDD-Tests für eine Umrechnungsfunktion. Prüfe korrekte Werte **und** Fehlerverhalten (ungültige Eingaben).

Feature-Datei (Gerüst + Hinweise)

Datei: features/temperature.feature

```
# HINWEIS: Verwende klare, messbare Erwartungen (z. B. exakte Zahlen).
```

Du kannst später Toleranzen im Code prüfen (z. B. `|result-expected| < 1e-6`).

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)

HINWEIS: Strings oder None sollen eine Exception auslösen
Given eine ungültige Eingabe "heiss"
When ich eine Umrechnung ausführe
Then wird ein ValueError ausgelöst

Schrittdefinitionen (Gerüst + Hinweise)

Datei: features/steps/temperature_steps.py

```
# HINWEIS: Importiere deine Produktionsfunktionen aus app.converter
# from app.converter import c_to_f, f_to_c

from behave import given, when, then

@given("eine Celsius-Temperatur von {c:d}")
def step_given_celsius(context, c):
    # TODO: im context speichern (z. B. context.input_c = c)
    pass

@given("eine Fahrenheit-Temperatur von {f:d}")
def step_given_fahrenheit(context, f):
    # TODO: im context speichern
    pass

@given('eine ungültige Eingabe "{text}"')
def step_given_invalid(context, text):
    # TODO: im context speichern
    pass

@when("ich nach Fahrenheit umrechne")
```

```
def step_when_to_f(context):
    # TODO: c_to_f(context.input_c) aufrufen und Ergebnis speichern
    # HINWEIS: beachte Datentypen (int/float)
    pass

@when("ich nach Celsius umrechne")
def step_when_to_c(context):
    # TODO: f_to_c(context.input_f) aufrufen und Ergebnis speichern
    pass

@when("ich eine Umrechnung ausführe")
def step_when_convert_any(context):
    # TODO: absichtlich falschen Typ übergeben und Exception im context
    merken
    # Tipp: try/except und die Exception unter context.exc ablegen
    pass

@then("erhalte ich {expected:float} Fahrenheit")
def step_then_f(context, expected):
    # TODO: numerischen Vergleich; Toleranz verwenden (z. B. 1e-6)
    pass

@then("erhalte ich {expected:float} Celsius")
def step_then_c(context, expected):
    # TODO: numerischen Vergleich
    pass

@then("wird ein ValueError ausgelöst")
def step_then_error(context):
    # TODO: prüfen, ob context.exc existiert und vom Typ ValueError ist
    pass
```

Produktivcode (Gerüst + Hinweise)

Datei: app/converter.py

```
# HINWEIS: Implementiere die beiden Funktionen gemäß Formeln:
#    $F = C * 9/5 + 32$ 
#    $C = (F - 32) * 5/9$ 
# Validiere Eingaben: akzeptiere nur int/float, sonst ValueError.

def c_to_f(celsius):
    # TODO: Eingabe validieren (isinstance)
    # TODO: Umrechnen und Rückgabe
    raise NotImplementedError("c_to_f ist noch nicht implementiert")

def f_to_c(fahrenheit):
    # TODO: Eingabe validieren (isinstance)
    # TODO: Umrechnen und Rückgabe
```

```
raise NotImplementedError("f_to_c ist noch nicht implementiert")
```

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

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

Feature-Datei (Gerüst + Hinweise)

Datei: features/login.feature

```
# HINWEIS: Nutze Background, um einen Startzustand für alle Szenarien zu definieren.

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 (Gerüst + Hinweise)

Datei: features/steps/login_steps.py

```
# HINWEIS: Importiere deinen AuthService aus app.auth
# from app.auth import AuthService

from behave import given, when, then

@given('es existiert ein Benutzer "{username}" mit Passwort "{password}"')
```

```
def step_given_user_exists(context, username, password):
    # TODO: AuthService im context instanzieren und Benutzer registrieren
    # HINWEIS: Leere Strings sollten in register() zu ValueError führen
    # (testen könnt ihr später erweitern)
    pass

@when('ich mich mit Benutzer "{username}" und Passwort "{password}"
anmelde')
def step_when_login(context, username, password):
    # TODO: login() aufrufen und Ergebnis (True/False) im context speichern
    pass

@then("bin ich eingeloggt")
def step_then_logged_in(context):
    # TODO: Ergebnis prüfen (True)
    pass

@then("ist der Login abgelehnt")
def step_then_denied(context):
    # TODO: Ergebnis prüfen (False)
    pass
```

Produktivcode (Gerüst + Hinweise)

Datei: app/auth.py

```
# HINWEIS: Minimaler AuthService ohne externe Abhängigkeiten.
# - _users als In-Memory-Dict
# - register(username, password): validiert Eingaben (nicht leer), speichert
Benutzer
# - login(username, password): True bei passender Kombination, sonst False

class AuthService:
    def __init__(self):
        # TODO: In-Memory-"Datenbank" initialisieren
        self._users = None # TODO: durch {} ersetzen

    def register(self, username, password):
        # TODO: Eingaben validieren (nicht leer), sonst ValueError
        # TODO: Benutzer speichern
        raise NotImplementedError("register ist noch nicht implementiert")

    def login(self, username, password):
        # TODO: Benutzer nachschlagen und Passwort vergleichen
        # Rückgabe: True bei Erfolg, sonst False
        raise NotImplementedError("login ist noch nicht implementiert")
```

Ausführung

Im Projekt-Hauptordner:

```
pip install behave
behave
```

Erwartung: Beide Features laufen grün, nachdem ihr die Schrittdefinitionen **und** den Produktivcode implementiert habt. **Optional:** Ergänzt weitere Szenarien (z. B. Passwort-Minimallänge, Whitespace-Trimming, Grenzwerte bei Temperaturen).

MXXX-LU08



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

Permanent link:
<https://wiki.bzz.ch/modul/m450/learningunits/lu16/aufgaben/behave?rev=1761200520>

Last update: **2025/10/23 08:22**

