Behaviour Driven Development (BDD) mit Behave

Einführung

Behaviour Driven Development (BDD) ist eine Weiterentwicklung von **Test Driven Development (TDD)**. Beim BDD steht das **Verhalten eines Systems aus Sicht der Benutzer:innen** im Zentrum. Tests werden in einer **natürlichen Sprache** formuliert, die von allen verstanden wird — nicht nur von Programmierer:innen.

Ein populäres Framework für BDD in Python ist **behave**. Damit lassen sich **lesbare Akzeptanztests** in einer sogenannten **Gherkin-Syntax** schreiben und mit Python-Code verknüpfen.

Ziel von BDD

- Verständliche Tests schreiben, die auch Fachpersonen lesen können
- Anforderungen und Tests in einer gemeinsamen Sprache beschreiben
- Entwicklungsschritte von der Benutzerstory bis zum Test automatisieren
- Früh Feedback erhalten, ob das System das gewünschte Verhalten zeigt

---

Aufbau eines Behave-Projekts

Ein Behave-Projekt besteht aus folgenden Ordnern und Dateien:

```
project/
├── features/
│   ├── calculator.feature
│   └── steps/
│       └── calculator_steps.py
```

- **`.feature`-Dateien** enthalten Szenarien in Gherkin-Sprache
- **`.py`-Dateien** im Unterordner `steps/` enthalten die Python-Schritte dazu

---

Gherkin-Syntax

Gherkin ist eine einfache Sprache, um Verhalten zu beschreiben. Jede Datei beschreibt ein **Feature** (eine Funktion oder Benutzerstory).

Beispiel:

```gherkin Feature: Einfacher Taschenrechner

```
Um einfache Rechnungen zu machen
Möchte ich zwei Zahlen addieren können
```

```
Scenario: Zwei Zahlen addieren
  Given ich habe 50 in den Rechner eingegeben
  And ich habe 70 in den Rechner eingegeben
  When ich drücke addieren
  Then sollte das Ergebnis 120 auf dem Bildschirm stehen
```

```

**Schlüsselwörter:**

- `Feature` → beschreibt die Funktion / das Ziel
- `Scenario` → einzelner Testfall
- `Given` → Ausgangslage (Vorbedingungen)
- `When` → Aktion (was passiert)
- `Then` → erwartetes Ergebnis
- `And` / `But` → Erweiterungen

---

# Schrittdefinitionen (Step Definitions)

Jeder Schritt im Feature wird mit einer Python-Funktion verknüpft.

```python from behave import given, when, then

@given('ich habe {zahl:d} in den Rechner eingegeben') def step_eingabe(context, zahl):

```
  if not hasattr(context, "zahlen"):
      context.zahlen = []
  context.zahlen.append(zahl)
```

@when('ich drücke addieren') def step_add(context):

```
  context.resultat = sum(context.zahlen)
```
```

@then('sollte das Ergebnis {erwartet:d} auf dem Bildschirm stehen') def step_pruefen(context, erwartet):

```
  assert context.resultat == erwartet
```

```

**Wichtig:**

- Der `context` ist ein gemeinsames Objekt für alle Schritte eines Szenarios
- Parameter in geschweiften Klammern `{zahl:d}` werden automatisch geparst (z. B. als int)

---

# Tests ausführen

Im Terminal im Projektordner:

```
' behave '
```

Ergebnis:

```
' Feature: Einfacher Taschenrechner Scenario: Zwei Zahlen addieren Given ich habe 50 in den Rechner eingegeben And ich habe 70 in den Rechner eingegeben When ich drücke addieren Then sollte das Ergebnis 120 auf dem Bildschirm stehen 1 feature passed, 0 failed '
```

Vorteile von Behave

- Tests sind **leicht lesbar** für alle Beteiligten
- **Klarer Bezug** zu Anforderungen / User Stories
- **Wiederverwendbare Schritte** in Python
- **Automatisierbar** in CI/CD-Pipelines (z. B. GitHub Actions, Jenkins)

Vergleich zu Unit Tests

Aspekt	Unit Test	BDD (Behave)
———	—————	—————
Fokus	einzelne Funktion / Klasse	Verhalten aus Nutzersicht
Sprache	Code (Python)	Gherkin (natürliche Sprache)
Zielgruppe	Entwickler:innen	Fachpersonen + Entwickler:innen
Beispiel	`assert add(2,3)==5`	„Given ich habe 2 und 3, When ich addiere, Then erhalte ich 5“

```

Installation

Installation mit `pip`:

``` pip install behave ```

Typische Fehlerquellen

- Schrittdefinition fehlt oder Schreibweise weicht ab
- Python-Datei liegt nicht im `steps/`-Ordner
- Context-Variable nicht initialisiert
- Feature-Datei hat keine Endung `.feature`

Weiterführende Links

- Offizielle Dokumentation: https://behave.readthedocs.io
- Gherkin-Syntax Referenz: https://cucumber.io/docs/gherkin/reference/
- Vergleich zu pytest-bdd: https://pytest-bdd.readthedocs.io

Aufgabe (optional)

Erstelle ein eigenes Feature:

```
1. Erstelle eine Datei ''temperature.feature''
2. Beschreibe ein Verhalten zum Umrechnen von Celsius in Fahrenheit
3. Implementiere die Schrittdefinitionen in Python
4. Führe die Tests mit ''behave'' aus
```
```