LU03b - PyTest

PyTest ist ein Test-Framework basierend auf Python. Es eignet sich vor allem, um die Schnittstellen einzelner Funktionen zu testen.

Einstieg

Die Nutzung von PyTest in unserer IDE PyCharm ist relativ einfach.

Paket "pytest" installieren

Zuerst musst du das Paket `pytest` im Virtual Environment deines Projekts installieren. Gehe in den Tab `Terminal` von PyCharm. Achte auf die Anzeige `(venv) …` am Anfang des Prompts.

```
(venv) PS C:\BZZ\Python\MyProject>
```

Falls diese Anzeige fehlt, musst du die virtuelle Umgebung zuerst aktivieren. Gib dazu unter Windows den Befehl `.\venv\Scripts\activate.ps1` ein.

Nun kannst du pytest installieren.

```
(venv) PS C:\BZZ\Python\MyProject> pip3 install pytest
```

Ich empfehle dir, die Pakete immer im Virtual Environment zu installieren. Dadurch gibt es weniger Probleme mit der Kompatiblität verschiedener Projekte.

Modul für Testfälle

Nachdem pytest installiert ist, erstellst du ein neues Modul (Python File) für deine Tests. Als Dateinamen ergänze ich den Namen des zu testenden Moduls (z.B. main.py) um `test_` (z.B. test_main.py).

Unit Tests

Die einzelnen Unit Tests werden als Funktionen in Python programmiert. Zum Beispiel:

```
def test_normal():
    result = factorial(7)
    assert result == 5040
```

- Der Funktionsname für den Test muss mit `test_` beginnen. Andernfalls erkennt `pytest` die Funktion nicht.
- Wir rufen die zu testende Funktion (im Beispiel `factorial`) mit den Testdaten auf.
- Der Befehl `assert` vergleicht das tatsächliche Resultat mit dem erwarteten Resultat.

Assert

Der Befehl `assert` ist eine spezielle Bedingung, die wir zum Testen und Debuggen von Code verwenden. Falls die Bedingung erfüllt ist, wird nichts gemacht. Sonst wird eine `AssertionError`-Exception geworfen. Wir könnten das gleiche Resultat auch mit `if/else` erreichen:

assert	**if / else**

`assert result == 5040`	`if result == 5040:`
	`  pass`
	`else:`
	`  raise AssertionError`

Beispiel

Anhand dieses einfachen Beispiels siehst du, wie eine Python-Funktion getestet werden kann.

factorial.py

Dieses Modul enthält eine Funktion um die Fakultät einer Zahl zu berechnen.

```
def factorial(number):
    fact = 1
    for count in range(1, number + 1):
        fact = fact * count
    return fact
```

test_factorial.py

Dieses Modul enhält meine Unit Tests.

```
from factorial import factorial


def test_normal():
    result = factorial(7)
    assert result == 5040

def test_zero():
    result = factorial(0)
    assert result == 1
```

Resultate mit Fliesskommazahlen

Bei Berechnungen mit Fliesskommazahlen gibt es immer kleine Abweichungen bei den Resultaten. Daher würde der Befehl `assert` häufig Fehler melden, obwohl das Resultat grundsätzlich korrekt ist.

Um diesem Problem zu begegnen, gibt es die Funktion ''pytest.approx()''. Diese Funktion vergleicht die Werte unter Berücksichtigung einer geringen Toleranz. Beim Aufruf der Funktion muss zwingend das erwartete Resultat mitgegeben werden. Zusätzlich kennt die Funktion 3 optionale Argumente:

- `rel`: Erlaubte relative Abweichung. Default: 1/1'000'000
- `abs`: Erlaubte absolute Abweichung. Default: 1/1'000'000'000
- `nan_ok`: Soll `NotANumber` erlaubt sein. Default: False

Beispiel

In diesem Beispiel testen wir eine Funktion, welche die Mehrwertsteuer (engl. **V**alue **A**dded **T**ax) berechnet. Wir haben ein Total von CHF 1575.50 was gemäss Taschenrechner einer Steuer von CHF 127.6155 entspricht.

```
import pytest

def calculate_vat(amount):
   return amount * 0.081

def test_calculate_vat():
    total = 1575.50
    vat = calculate_vat(total)
    pytest.approx(127.6155) == vat
```

Tutorials

- Tutorialpoints
- Youtube

M450-LU03