

---

# Softwareentwicklung

## Test Driven Development mit Python

---

SEW  
5AHITT 2016/17

Hauer Miriam

Note:  
Betreuer: Dominik Dolezal

Version 0.2  
Begonnen am 24. Oktober 2016  
Beendet am 24. Oktober 2016

# Inhaltsverzeichnis

<b>1</b>	<b>Aufgabenstellung</b>	<b>1</b>
<b>2</b>	<b>Ergebnisse</b>	<b>2</b>
2.1	Sphinx Installieren . . . . .	2
2.2	Sphinx Quickstart . . . . .	2
2.3	Run Config erstellen . . . . .	2
2.4	Vorgehensweise . . . . .	2
2.5	Programmierung . . . . .	3
<b>3</b>	<b>Probleme</b>	<b>6</b>
3.1	Dauer . . . . .	6

# 1 Aufgabenstellung

Schreiben Sie zu die Klasse Bruch in einem Modul bruch

Nutzen Sie die Testklassen in PyCharm.

Ziel: Coverage > 95

Empfohlene Vorgehensweise:

- Projekt in PyCharm erstellen
- Modul bruch erstellen
- Klasse Bruch erstellen
- Test-Ordner erstellen
- Unit-Tests entpacken und lauffähig machen

Abgabe:

Protokoll mit Testreports (inkl. Coverage) und Dokumentation (html)

Abgabe des Python-files

Achtung: Vergessen Sie nicht auf eine ausführliche Dokumentation mittels sphinx

## 2 Ergebnisse

### 2.1 Sphinx Installieren

Um Sphinx verwenden zu können wird ein Pythoninterpreter vorausgesetzt. Da dies bereits zu erledigen und daher nicht Teil dieser Aufgabe war werde ich darauf nicht weiter eingehen. Hat man den Interpreter installiert werden mit den Folgenden Kommandos in der cmd pip und sphinx installiert.

```
1 python get-pip.py
   pip install sphinx
```

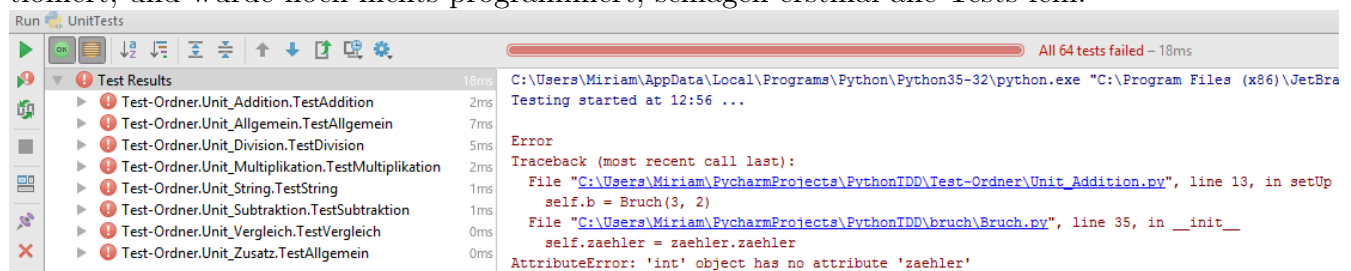
Hierbei installiert man mittels “python get-pip.py” das Kommando “pip” um es in der zweiten Zeile, zum Download und Installieren von sphinx, verwenden zu können.

### 2.2 Sphinx Quickstart

Danach wird Sphinx Quickstart wie in der zur Verfügung gestellten pdf zu sehen, geöffnet. Im Gegensatz zu den Unterlagen wird aber kein neues Fenster geöffnet sondern es startet ein Script mit einigen Abfragen (path,namen,etc,..) in der PyCharm Konsole welche man nach persönlicher Referenz beantworten kann.

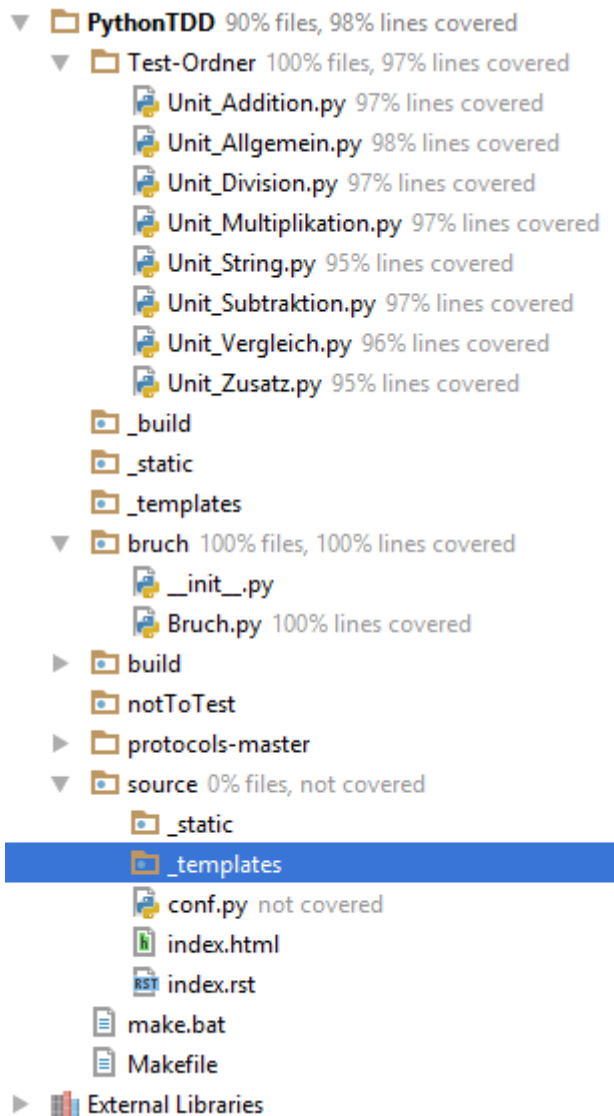
### 2.3 Run Config erstellen

Um eine neue Run-Config zu erstellen wird der Button rechts oben in pycharm angeklickt. Dieser führt zu dem in der pdf zu sehenden Fenster in dem man nun den Folder und das Pattern angibt. AuSSerdem kann ausgewählt werden ob man alle Klassen in dem Ordner, nur ein bestimmtes Script, eine Klasse, eine Methode, etc., ausführen möchte. Nachdem die neuen Einstellungen applied wurden kann man unter dem gewählten Namen die Testklasse(n) ausführen. Hat alles funktioniert, und wurde noch nichts programmiert, schlagen erstmal alle Tests fehl.



### 2.4 Vorgehensweise

Um die Übersicht zu wahren wurden die einzelnen Testklassen anstatt der Testall-Klasse verwendet. Statt des einen empfohlenen Test Ordners ist ein zweiter vorhanden mit Test-Klassen die momentan nicht von Relevanz sind.



## 2.5 Programmierung

Um die gegebenen Testfälle zu bestehen mussten folgende Methoden überladen werden um sie an das Rechnen mit Brüchen anzupassen:

- `__eq__`
- `__float__`
- `__int__`
- `__complex__`
- `__invert__`
- `__str__`

- `__abs__`
- `__neg__`
- `__pow__`
- `__add__`
- `__iadd__`
- `__radd__`
- `__truediv__`
- `__rtruediv__`
- `__itruediv__`
- `__mul__`
- `__rmul__`
- `__imul__`
- `__sub__`
- `__isub__`
- `__rsub__`
- `__ge__`
- `__le__`
- `__lt__`
- `__gt__`
- `__iter__`

Sie erfüllen prinzipiell noch die selben Aufgaben, können diese aber konform der Bruchrechenregeln erarbeiten. Es folgt eine Methode zur Veranschaulichung.

```

2  def __add__(self, other):
    """
    Überschreibt die Methode add, sodass ein Bruch zurück gegeben wird der nach den Bruchrechenregeln
    eine korrekte Summe zweier Brüche darstellt.
4  :param other: Bruch der addiert wird
    :return: neuen Bruch = addition von self und other
    """
6  if isinstance(other, (float, str)):
8      raise TypeError
    other = Bruch(other)
10 return Bruch(self.zaehler * other.nenner + other.zaehler * self.nenner, self.nenner * other.nenner
    )

```

Listing 1: add methode

Zusätzlich wurde ein Konstruktor und eine `_Bruch_makeBruch` Methode geschrieben.

```

1  def __init__(self, zaehler, nenner=None):
2      """
3      __init__ nimmt als Konstruktor einen zähler und einen nenner, mit dem standard-wert none,
4      entgegen.
5      danach wird die Validität der einzelnen Parameter überprüft und gegebenenfalls (nenner = 0, nenner
6      oder zähler ist float)
7      eine Exception geworfen. falls nicht 2 int-werte sondern nur 1 bruch objekt übergeben wurde bleibt
8      der nenner none und
9      es werden zähler und nenner aus dem objekt eingelesen.
10     wird bloß 1 int wert übergeben wird der nenner auf 1 gesetzt (ganze zahl)
11
12     :param zaehler: zähler bzw ganze zahl
13     :param nenner:  nenner bzw none
14     """
15     if nenner == 0:
16         raise ZeroDivisionError ("zerodiverr")
17     if isinstance(nenner, float) :
18         raise TypeError
19     if isinstance(zaehler, float):
20         raise TypeError
21     if (nenner != None):
22         self.zaehler = zaehler
23         self.nenner = nenner
24     elif nenner == None and isinstance(zaehler, int):
25         self.zaehler = zaehler
26         self.nenner = 1
27     else :
28         self.zaehler = zaehler.zaehler
29         self.nenner = zaehler.nenner

```

Listing 2: Konstruktor

```

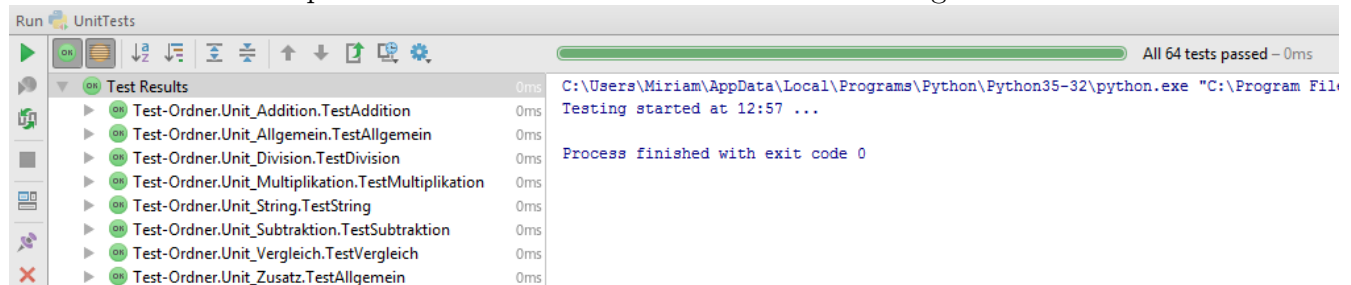
1  def __Bruch__makeBruch(value):
2      """
3      private Methode die einen Bruch über die Eingabe eines einzelnen Wert erstellen lässt. Wirft einen
4      TypeError, falls ein String übergeben wird
5      :return: neuen Bruch
6      """
7      if isinstance(value, str):
8          raise TypeError
9      else:
10         return Bruch(value, value)

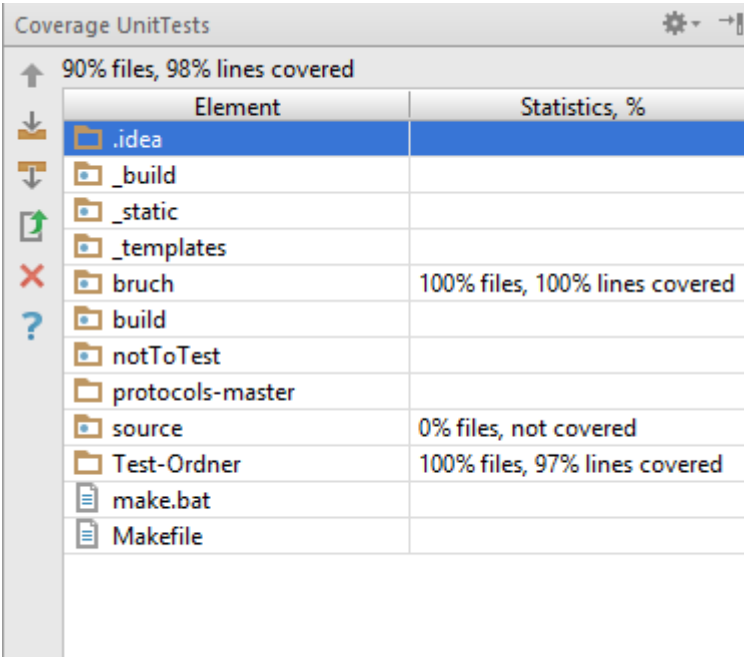
```

Listing 3: `_Bruch_makeBruch`

Nach der Implementieren funktionieren alle Testfälle. Wobei die Codecoverage des Bruch.py Files 100 Prozent beträgt.

Daraufhin wird die Sphinx-Dokumentation mittels ausführen der generierten make.bat erstellt.





Coverage UnitTests

90% files, 98% lines covered

Element	Statistics, %
.idea	
_build	
_static	
_templates	
bruch	100% files, 100% lines covered
build	
notToTest	
protocols-master	
source	0% files, not covered
Test-Ordner	100% files, 97% lines covered
make.bat	
Makefile	

### 3 Probleme

Bei der eigentlichen Programmierung sind keine Probleme aufgetreten. Die Verwendung von Sphinx hat Anfangs allerdings einige Schwierigkeiten verursacht. Zunächst wurden einige Errors beim Öffnen angezeigt sobald man das gebildete File verwenden wollte, was auf einen Fehler beim Builden zurückzuführen war. Es wurde zuerst versucht die Dokumentation über PyCharm zu erstellen, was sich als schlechte Idee heraus gestellt hat. Nach einer kurzen Internetrecherche hat sich herausgestellt, dass das make.bat-File ausgeführt werden muss um zu Builden.

Nachdem dieser Teil funktionierte, fand er ein Modul nicht weil ich vergessen hatte den package namen davor anzugeben, nach dieser Beifügung lief Sphinx problemlos.

#### 3.1 Dauer

Insgesamt hat diese Arbeit ungefähr 5 Stunden in Anspruch genommen. Davon sind 3.5 Stunden Programmier-Arbeitszeit und 1.5 Stunden für Dokumentation.



## Literatur

## Tabellenverzeichnis

## Listings

1	add methode . . . . .	4
2	Konstruktor . . . . .	5
3	__Bruch__makeBruch . . . . .	5

## Abbildungsverzeichnis