A04: TDD Bruch

Implementierung aller mathematischer und programmiersprachlicher Umsetzungen für das Rechnen mit Brüchen

Raphael Simsek 5AHITM

A04: TDD Bruch

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

Vorbereitung

geschätzte Zeit: 180 Minuten *tatsächliche Zeit: 90 Minuten*

Programmierung

Um mich auf den Aufwand der Programmierung vorzubereiten schob ich zuerst die komplette Aufgabe viel zu weit in die Ferne, wodurch ich diesen Rückstand dann nur sehr schwer aufholen konnte. Dieses Aufschieben war durch eine meinen verlangsamten Einstieg in die fünfte Klasse, aufgrund meiner vorgezogenen Matura und der darauffolgenden Aufholzeit zu begründen.

Als ich dann endlich begonnen habe mir Gedanken über das Beispiel zu machen, habe ich durch Befragung meiner Kollegen heraus gefunden dass das Beispiel vor allem mittels der *predefined* Funktionen aus Python umzusetzen sind, wodurch ich im Bezug auf diese begann zu recherchieren.

Bald konnte ich wertige Informationen zum Beispiel hier finden, wo mir aus der Auflistung aller vordefinierten Funktionen schnell erkennen konnten welche für die Verwendung einer Bruchrechnung essenziell sind.

Sphinx

Es war mir sehr leicht möglich mittels der Sphinx Dokumentation herauszufinden wie dieses installiert werden muss und auch die ersten Schritte mit Sphinx ließen mich für diesen Teil der Aufgabenstellung zuversichtlich werden.

pip3 install –U Sphinx *sphinx-quickstart*

Lessons learned:

Zukünftige SEW Aufgaben früher beginnen, um besser komplett eigene Erfahrungen bei der Informationsaquirierung sammeln zu können. Zwischenabgaben und Fortschrittskontrollen sind in Zukunft von mir einzuhalten um laufenden Fortschritt zu erzwingen.

Zu Python an sich in keiner Reihung:

- Python ist toll lesbare Programmiersprache.
- Python Unit Testing ist angenehm aufgebaut und man kann sich die meisten Fehler einfach mittels der Tests selbst herleiten und diese ausbessern

Zu Sphinx:

- Ist leicht zu installieren.
- Verfügt über einigermaßen empfehlbare Dokumentation

Python - Programmierung

geschätzte Zeit: 480 Minuten *tatsächliche Zeit: 890 Minuten*

Methoden Findung

Wie bereits in der Vorbereitung erwähnt habe ich sehr schnell durch meine Klassenkollegen erfahren dass ich die vordefinierten Methoden von Python überschreiben muss, wodurch ich natürlich sehr zielgerichtet gesucht habe.

Aus meiner Suche ergab sich dass folgende Methoden überschrieben werden musste um Rechenoperationen und programmiersprachliche Operationen erfolgreich ausführen zu können:

- **__iter__**
  Mit dem Iterator wurde die Iterierung abgeändert um diese mittels einem Bruch zu ermöglichen um beispielsweise dieses Code Snippet durchführen zu können:
  self.zaehler, self.nenner = zaehler
- **__init__**
  Mittels dieser Methode wird der Konstruktor überschrieben um durch diesen die gelieferten Parameter bereits bei der Erzeugung der Klasse auf Fehler durchsuchen zu können, so werden beispielsweise Nenner = 0 innerhalb des Konstruktors ausgeschlossen
- **__float__**
  Aufgrund der Überschreibung dieser Methode ist es möglich zu jedem Zeitpunkt mittels der *float()* Methode einen aktuellen Float-Wert eines Bruch-Objekts zu erhalten. Da future division hier nicht importiert wurde muss die Umsetzung direkt mittels der Division passieren. *float(Bruch)*
- **__int__**
  Da diese Methode überschrieben wird kann zu jedem Zeitpunkt ein aktueller Int-Wert eines Bruch-Objektes mittels *int()* erlangt werden. *int(Bruch)*
- **__invert__**
  Aufgrund der Überschreibung dieser Methode kann man sehr einfach mittels einer Tilde (~) einen Bruch invertieren, wodurch diese speziell für Brüche adaptiert werden musste. *~Bruch*
- **__neg__**
  Durch die Überschreibung der Negation kann man jeden Bruch mittels eines Minus negieren. *-Bruch*
- **__pow__**
  Durch die Adaptierung dieser Methode können auch Brüche mit einer Hochzah (int) mittels ** versehen werden. Brüche mit einer Hochzahl werden mittels Zähler ^ Zahl und Nenner ^ Zahl errechnet. *Bruch ** 3*
- **__abs__**

Durch das Überschreiben dieser Methode wurde das Erhalten des Absolutbetrags eines Bruches ermöglicht. *abs(Bruch)*

- **__eq__**
  Das Überschreiben von equals ermöglicht es zu überprüfen ob zwei Brüche ident sind. *Bruch1 == Bruch2*
- **__ne__**
  Das Überschreiben von not equals ermöglicht es zu überprüfen ob zwei Brüche nicht ident sind. *Bruch1 != Bruch2*
- **__ge__**
  Das Überschreiben von greater equals ermöglicht es zu überprüfen ob der erste Bruch größer oder gleich dem anderen Bruch ist. *Bruch1 >= Bruch2*
- **__gt__**
  Das Überschreiben von greater ermöglicht es zu überprüfen ob der erste Bruch größer dem anderen Bruch ist. *Bruch1 > Bruch2*
- **__le__**
  Das Überschreiben von lower equals ermöglicht es zu überprüfen ob der erste Bruch kleiner oder gleich dem anderen Bruch ist. *Bruch1 <= Bruch2*
- **__lt__**
  Das Überschreiben von lower ermöglicht es zu überprüfen ob der erste Bruch kleiner dem anderen Bruch ist. *Bruch1 < Bruch2*
- **__repr__**
  Das Überschreiben der Repräsentation Methode ermöglicht mir, ähnlich wie es das Überschreiben der String (str()) Methode tun würde, die String Values jedes Bruches festzusetzen und die Ausgabe eines Bruches als String zu bestimmen. *str(Bruch)*
- **__add__**
  Das Überschreiben von der Additionsmethode ermöglicht mir zwei Brüche zu addieren. *Bruch1 + Bruch2*
- **__radd__**
  Das Überschreiben von der rechten Additionsmethode ermöglicht mir zwei Brüche zu addieren, bei denen die Addition verkehrt gespiegelt zur ersten Methode umgesetzt wird . *Bruch2 + Bruch1*
- **__iadd__**
  Das Überschreiben der in-place Addition ermöglicht mir schneller Additionen bei denen ich das in der Rechnung involvierte Element überschreiben möchte umzusetzen. Dieses Modell ist auch bereits aus Java bekannt.
  Bruch1 += Bruch2
- **__sub__**
  Das Überschreiben von der Subtraktionsmethode ermöglicht mir zwei Brüche zu subtrahieren. *Bruch1 - Bruch2*
- **__rsub__**
  Das Überschreiben von der rechten Subtraktionsmethode ermöglicht mir zwei Brüche zu subtrahieren, bei denen die Addition verkehrt gespiegelt

zur ersten Methode umgesetzt wird. Hier ergibt sich ein Problem das Kommutativgesetz für Subtraktionen nicht einsetzbar ist. *Bruch2 - Bruch1*

- **__isub__**
  Das Überschreiben der in-place Subtraktion ermöglicht mir schneller Subtraktionen bei denen ich das in der Rechnung involvierte Element überschreiben möchte. Dieses Modell ist auch bereits aus Java bekannt. *Bruch1 -= Bruch2*
- **__mul__**
- Das Überschreiben von der Multiplikationsmethode ermöglicht mir zwei Brüche zu multiplizieren. *Bruch1 * Bruch2*
- **__rmul__**
- Das Überschreiben von der rechten Multiplikationsmethode ermöglicht mir zwei Brüche zu multiplizieren, bei denen die Multiplikation verkehrt gespiegelt zur ersten Methode umgesetzt wird. Hierfür gilt hier Zähler * Zähler & Nenner * Nenner.
  *Bruch2 * Bruch1*
- **__imul__**
- Das Überschreiben der in-place Multiplikation ermöglicht mir schneller Multiplikationen bei denen ich das in der Rechnung involvierte Element überschreiben möchte umzusetzen. Dieses Modell ist auch bereits aus Java bekannt. *Bruch1 *= Bruch2*
- **__truediv__**
  Das Überschreiben von der Divisionsmethode ermöglicht mir zwei Brüche zu dividieren, was einen klassischen Doppelbruch, also außen Mal außen durch innen Mal innen, darstellt. *Bruch1 / Bruch2*
- **__rtruediv__**
  Das Überschreiben von der rechten Division ermöglicht mir zwei Brüche zu dividieren, bei denen die Division umgekehrt zu truediv umgesetzt wird. Hier ergibt sich ein Problem das Kommutativgesetz für Divisionen nicht einsetzbar ist. *Bruch2 / Bruch1*
- **__itruediv__**
  Das Überschreiben der in-place Division ermöglicht mir schneller Divisionen bei denen ich das in der Rechnung involvierte Element überschreiben möchte. Dieses Modell ist auch bereits aus Java bekannt. *Bruch1 /= Bruch2*

Folgende Methode muss laut der Testfälle ebenfalls erstellt werden:

__makeBruch:

Erstellt eine statische nicht vordefinierte Funktion welche aus einer int Zahl einen Bruch macht und diesen zurück gibt.

Sphinx (Python Dokumentationstool):

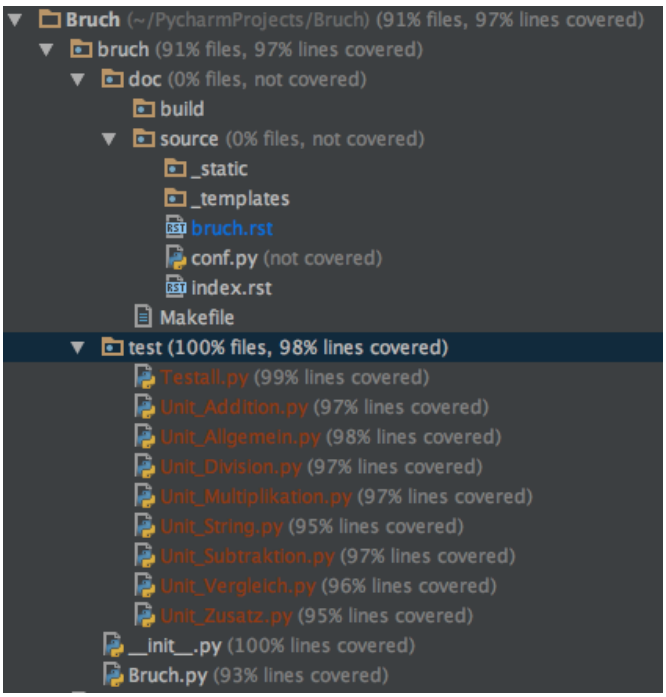
Sphinx ließ sich mittels dem pip-Befehl sehr leicht installieren, allerdings traf ich bereits hier auf Probleme in welchen Interpreter ich Sphinx installieren konnte.

Der Quickstart funktionierte problemlos, wobei am Wichtigsten die Trennung von source und build laut Ihrem Sphinx-PDF ist. Auch die Erstellung und Änderung von Index.rst und Bruch.rst fiel mir durch das PDF leicht.

Bei der Generierung stieß ich jedoch an eine undurchdringbare Wand an, wodurch mir hierfür mein Kollege Adaresh Soni half, da sein Sphinx richtig auf Systemebene ist.

Test – Ergebnisse:

Anhand der Screenshots kann man erkennen dass ich leider in meinem Bruch.py File die 95% Coverage nicht ganz erreicht habe, allerdings ist es mir wirklich ein Rätsel wieso dies der Fall ist, denn alle Zeilen die noch als nicht in Verwendung angezeigt werden sind returns und diverse andere Befehle, welche keinesfalls ersatzlos ausgetauscht werden können. Jedoch wurde jeder Test positiv abgeschlossen, wie Sie dem Test Results.html File entnehmen können.



Lessons learned

Positiva:

- Python ist eine tolle Programmiersprache.
- Ich habe gelernt was für eine große Bandbreite an vordefinierten Methoden man bei Python überschreiben kann.
- Umgang mit PyCharm
- Umgang mit Unit Tests in Python
- Installation von Sphinx
- Sphinx lässt sich leicht für die benötigten Methoden konfigurieren

Negativa:

- Die mathematischen Funktionen, allen voran rtruediv, sub, rsub und isub, waren für mich syntaktisch sehr schwierig funktionstüchtig zu implementieren → bitte kein fächerübergreifendes Überlegen voraussetzen
- Ist für mich nicht generierbar gewesen → Downgrade der Babel Version wäre nötig gewesen → Daher Hilfe von Adaresh Soni erhalten

view all my commits on GitHub at: https://github.com/rsimsek-tgm/Bruch