A02 Bruch

Eren Sefer 5YHITM

Aufgabenstellung

Schreiben Sie zu die Klasse Bruch in einem Modul bruch

Nutzen Sie die Testklassen in PyCharm.
Ziel: Coverage > 95%

Empfohlene Vorgehensweise:

1. Projekt in PyCharm erstellen
2. Modul bruch erstellen
3. Klasse Bruch erstellen
4. Test-Ordner erstellen
5. 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

Arbeit vor dem Coden

Vor der eigentlichen Arbeit habe ich mir den Aufwand gemacht alle Testfälle zu analysieren. Ich habe mir aus allen magicfunctions die benötigten herausgesucht und in der folgenden Tabelle dokumentiert. Eine eigene Tabelle für alle Errors wurde zusätzlich erstellt.

Klassenname	Testfunktion	Beschreibung
__float__	testFloat	Damit ein float(bruch(1,1)) gemacht werden kann.
__int__	testInt	Damit ein int(bruch(1,1)) gemacht werden kann.
__complex__	testComplex	Damit ein complex(bruch(1,1)) gemacht werden kann.
__invert__	testInvert	Damit ~bruch(1,1) gemacht werden kann.
__repr__	testInteger	Damit bruch(3,1) als String (3) angezeigt wird
__pow__	testPow	Wenn Bruch aus ints besteht dann kann zaehler und nenner zur power von p gerechnet werden
__abs__	testabs	Damit ein abs(bruch(1,1) gemacht werden kann.

__neg__	testNeg	Damit ein -bruch(1,1) funktioniert.
	testRef1	Sollte von alleine funktionieren?
	testRef2	Sollte von alleine funktionieren?
__eq__	testEqual	Damit == funktioniert
__ne__	testNotEqual	Damit != funktioniert
__ge__	testGE	Damit >= funktioniert
__le__	testLE	Damit <= funktioniert
__lt__	testLT	Damit < funktioniert
__gt__	testGT	Damit > funktioniert
__repr__	testStr	Damit Bruch (3,2) als String (3,2) ausgegeben wird.
__repr__	testStr2	Damit Bruch (-3,-2) als String (3,2) ausgegeben wird.
__add__	testPlus	bruch(3,2) + bruch(4,5)
__add__	testPlus2	bruch + bruch
__add__	testPlus3	bruch + int
__radd_	testradd	int + bruch
__iadd__	testiAdd	"+=" int
__iadd__	testiAdd2	"+=" bruch

Errorname	Testfunktion	Beschreibung
ZeroDivisionError	testcreateBruchZeroError	Es wird gecheckt ob ein Bruch durch 0 erstellt werden kann.
IncompatibleTypeError	testcreateBruchWrongTypeNenner	Es wird gecheckt ob ein Bruch im Nenner ein anderer Datentyp als im Zähler hat
IncompatibleTypeError	testcreateBruchWrongTypeZaehler	Es wird gecheckt ob ein Bruch im Zaehler einen anderen Datentyp als im Nenner hat
IncompatibleTypeError	testPowError1	Es wird gecheckt ob man den Bruch mit einem Float hochrechnen kann
IncompatibleTypeError	testPowError2	Es wird gecheckt ob man den Bruch mit einem String hochrechnen kann
IncompatibleTypeError	test_makeBruchTypeError	Es wird gecheckt ob man einen Bruch mit einem String als zaehler und nenner erstellen kann

Code + Beispiel Anwendung

```
def __init__(self,zaehler=1,nenner=1):
        """
        Der Konstruktor

        :raises: TypeError: Falscher Datentyp
        :raises: ZeroDivisionError: 0 wurde für den Nenner gewählt.
        :param zaehler: Die Zahl beim Bruch oben als int oder Bruch
        :param nenner: Die Zahl beim Bruch unten als int
        """
        if nenner==0:
            raise ZeroDivisionError

        if type(zaehler) is int and type(nenner) is int:
            self.zaehler = zaehler
            self.nenner = nenner
        elif isinstance(zaehler, Bruch):
            self.zaehler = zaehler.zaehler
            self.nenner = zaehler.nenner
            return
        if type(zaehler) is not int or isinstance(zaehler,Bruch): raise
            TypeError('Wrong Datatype:'+type(zaehler).__name__+'!
Pleas use int or Bruch instead') if
        type(nenner) is not int:
            raise TypeError('Wrong Datatype:'+type(nenner).__name__+'!
Pleas use int instead')
```

Dies ist der Konstruktor. Er wird gebraucht um ein Objekt der Klasse zu erzeugen. Es folgt das Beispiel der Erzeugung einer Klasse.

```
b = Bruch(3,2)
```

```
def __float__(self):
        """
        ueberschreibt float()

        :return: float
        """
        back = float(self.zaehler/self.nenner)
        return back
```

Dies ist die Funktion float welche verwendet wird um einen Bruch in ein Float umzuwandeln. Es folgt das Beispiel für die Verwendung von Float.

```
b = Bruch(3,2
f = float(b)
```

```
def __int__(self):
        """
        ueberschreibt int()

        :return: int
        """
        back = int(self.__float__())
        return back
```

Die ist die Funktion int welche verwendet wird um einen Bruch in einen Int umzuwandeln. Es folgt das Beispiel für die Verwendung von Int.

```
b = Bruch(3,2)
i = int(b)
```

```
def __complex__(self):
        """
        ueberschreibt complex()

        :return: complex
        """
        back = complex(self.__float__())
        return back
```

Dies ist die Funktion complex welche verwendet wird um einen Bruch in eine Komplexe Zahl umzuwandeln Es folgt das Beispiel für die Verwendung von Complex.

```
b = Bruch(3,2)
c = complex(b)
```

```
def __invert__(self):
        """
        Einen Bruch invertieren

        :return: Bruch
        """
        back = Bruch(self.nenner,self.zaehler)
        return back
```

Dies ist die Funktion Invert welche nennen und Zähler vertauscht. Es folgt das Beispiel für die Verwendung von invert.

```
b = Bruch
(3,2) in = ~b
```

```
def __repr__(self):
        """
        Die Representation eines Bruches in Form eines Strings

        :return str: Die Representation
        """

        if self.nenner<0:
            self.nenner*=-1
            self.zaehler*=-1

        if self.nenner==1:
            return "(%d)" %
        self.zaehler else:
            return "(%d/%d)" % (self.zaehler, self.nenner)
```

Dies ist die Funktion repr welche die Ausgabe des Bruches als String ermöglicht. Es folgt das Beispiel für die Verwendung von repr.

```
b = Bruch(3,2)
print Bruch(3,2)
```

```
def __pow__(self, power):
        """
        Potenzieren eines Bruches

        :raises: TypeError: Falscher
        Datentyp :param power: Der Exponent
        :return: Bruch
        """
        if type(power) is int:
            return Bruch(self.zaehler ** power, self.nenner **
        power) else:
            raise TypeError('Wrong
Datatype:'+type(power).__name__+'! Pleas use int instead')
```

Dies ist die Funktion pow welche einen Bruch potenziert. Es folgt das Beispiel für die Verwendung

```
b = Bruch(3,2)
p = b ** 2
```

```
def __abs__(self):
        """
        Der Absolutwert eines Bruches

        :return: Bruch
        """
        back = Bruch(abs(self.zaehler),abs(self.nenner))
        return back
```

Dies ist die Funktion abs welchen den Absolut Betrag des Bruches errechnet. Es folgt das Beispiel für die Verwendung von abs.

```
b = Bruch (3,2)
a = abs(b)
```

```
def __neg__(self):
        """
        Das negativ eines Bruches

        :return: Bruch
        """
        back = Bruch(-
        self.zaehler,self.nenner) return back
```

Dies ist die Funktion neg welche den negativen Wert eines Bruches errechnet. Es folgt das Beispiel für die Verwendung von neg.

```
b = Bruch
(3,2) n = -b
```

```
def __makeBruch(other):
        """
        Um sicher zu gehen das ein Bruch erstellt/verwendet wird

        :raises:      TypeError:      Falscher
        Datentyp  :param  other:  Der  Bruch
        oder die Zahl :return: Bruch
        """

        if isinstance(other,
            Bruch): return other
        elif type(other) is int:
            b=Bruch(other,1)
            return b
        else:
            raise TypeError('Wrong
Datatype:'+type(other).__name__+'! Please use int or a Bruch')
```

Dies ist die Funktion makeBruch welche sicherstellt dass das gewüschte Objekt ein Bruch ist bzw. wird. Es folgt das Beispiel für die Verwendung von makeBruch

```
b = Bruch(3,2)
m = Bruch.__makeBruch(b)
```

```
def __eq__(self, other):
        """
        Bruch ist gleich mit

        :param other: anderer Bruch oder
        Zahl :return: boolean
        """
        other = Bruch.__makeBruch(other)
        if self.zaehler == other.zaehler and self.nenner
== other.nenner:
            return True
        elif float(self)==float(other):
            return True
        else:
            return False
```

Dies ist die Funktion eq welche überprüft ob ein Bruch gleich mit einem anderen Bruch ist. Es folgt das Beispiel für die Verwendung von eq.

```
b = Bruch(3,2)
b2 = Bruch(3,2)
#somekind of branch or loop with : b == b2
```

```
def __ne__(self, other):
        """
        Bruch ist nicht gleich mit

        :param other: anderer Bruch oder
        Zahl :return: boolean
        """
        other = Bruch.__makeBruch(other)
        if self.zaehler != other.zaehler or self.nenner != other.nenner:
            return True
        else:
            return False
```

Dies ist die Funktion ne welche überprüft ob ein Bruch nicht gleich mit einem anderen Bruch ist. Es folgt das Beispiel für die Verwendung von ne.

```
b = Bruch(3,2)
b2 = Bruch(3,2)
#somekind of branch or loop with : b != b2
```

```
def __le__(self, other):
        """
        Bruch ist kleiner gleich als

        :param other: anderer Bruch oder
        Zahl :return: boolean
        """
        other = Bruch.__makeBruch(other)
        if self.__float__() <=
            float(other): return True
        else:
            return False
```

Dies ist die Funktion le welche überprüft ob ein Bruch gleich mit einem anderen Bruch oder kleiner ist ist. Es folgt das Beispiel für die Verwendung von le.

```
b = Bruch(3,2)
b2 = Bruch(3,2)
#somekind of branch or loop with : b <= b2
```

```
def __ge__(self, other):
        """
        Bruch ist groeßer gleich als

        :param other: anderer Bruch oder
        Zahl :return: boolean
        """
        other = Bruch.__makeBruch(other)

        if self.__float__() >=
            float(other): return True
        else:
            return False
```

Dies ist die Funktion ge welche überprüft ob ein Bruch gleich mit einem anderen Bruch oder größer ist. Es folgt das Beispiel für die Verwendung von ge.

```
b = Bruch(3,2)
b2 = Bruch(3,2)
#somekind of branch or loop with : b >= b2
```

```
def __lt__(self, other):
        """
        Bruch ist kleiner als

        :param other: anderer Bruch oder
        Zahl :return: boolean
        """
        other = Bruch.__makeBruch(other) if
        self.__float__() < float(other):
            return
        True else:
            return False
```

Dies ist die Funktion lt welche überprüft ob ein Bruch kleiner als ein Bruch ist. Es folgt das Beispiel für die Verwendung von lt.

```
b = Bruch(3,2)
b2 = Bruch(3,2)
#somekind of branch or loop with : b < b2
```

```
def __gt__(self, other):
        """
        Bruch ist groeßer als

        :param other: anderer Bruch oder
        Zahl :return: boolean
        """
        other = Bruch.__makeBruch(other) if
        self.__float__() > float(other):
            return
        True else:
            return False
```

Dies ist die Funktion gt welche überprüft ob ein Bruch größer als ein Bruch ist. Es folgt das Beispiel für die Verwendung von gt.

```
b = Bruch(3,2)
b2 = Bruch(3,2)
#somekind of branch or loop with : b > b2
```

```
def __add__(self, other):
        """
        Das addieren eines Bruches mit einem Bruch oder einer Zahl

        :raises: TypeError: Falscher Datentyp
        :param other: Ein Bruch oder eine Zahl
        :return: Bruch
```

```
        """
        if isinstance(other, Bruch):
            other = Bruch.__makeBruch(other) return
            Bruch(self.zaehler*other.nenner
+self.nenner*other.zaehler, self.nenner *
        other.nenner) elif type(other) is int:
            return Bruch(self.zaehler +
        (other*self.nenner),self.nenner) else:
            raise TypeError('Wrong
Datatype:'+type(other).__name__+'! Please use int or Bruch')
```

Dies ist die Funktion add welche den welche 2 Brüche addiert. Es folgt das Beispiel für die Verwendung von add.

```
b = Bruch(3,2)
b2 = Bruch(14,4)

a = b + b2
```

```
    def __radd__(self, other):
        """
        Das addieren einer Zahl mit einem Bruch

        :param other: eine
        Zahl :return: Bruch
        """
        return self.__add__(other)
```

Dies ist die Funktion radd welche den welche einen Bruch mit einer Zahl addiert in reversierter Reihenfolge. Es folgt das Beispiel für die Verwendung von radd.

```
b = Bruch(3,2)

a = 1 + b2
```

```
    def __iadd__(self, other):
        """
        Das interne addieren eines Bruches

        :param other: Bruch oder
        Zahl :return: Bruch
        """
        other = Bruch.__makeBruch(other)
        self = self + other
        return self
```

Dies ist die Funktion iadd welche den welche 2 Brüche bzw einen Bruch und eine Zahl intern addiert. Es folgt das Beispiel für die Verwendung von iadd.

```
b = Bruch(3,2)

b += 1
b += b
```

```
    def __sub__(self, other):
        """
        Das subtrahieren eines Bruches mit einem Bruch oder einer Zahl

        :raises: TypeError: Falscher Datentyp
        :param other: Ein Bruch oder eine Zahl
        :return: Bruch
        """
        if isinstance(other, Bruch):
            other = Bruch.__makeBruch(other) return
            Bruch(self.zaehler*other.nenner-
self.nenner*other.zaehler, self.nenner * other.nenner)
        elif type(other) is int:
            return Bruch(self.zaehler -
        (other*self.nenner),self.nenner) else:
            raise TypeError('Wrong
Datatype:'+type(other).__name__+'! Please use int or Bruch')
```

Dies ist die Funktion sub welche den welche 2 Brüche subtrahiert. Es folgt das Beispiel für die Verwendung von sub.

```
b  =  Bruch(3,2)
b2  =  Bruch(4,3)
s = b - b2
```

```
    def __isub__(self, other):
        """
        Das interne subtrahieren eines Bruches

        :param other: Bruch oder
        Zahl :return: Bruch
        """
        other = Bruch.__makeBruch(other)
        self = self - other
        return self
```

Dies ist die Funktion isub welche den welche 2 Brüche bzw einen Bruch und eine Zahl intern subtrahiert. Es folgt das Beispiel für die Verwendung von isub.

```
b = Bruch(3,2)

b -= 1
b -= b
```

```
    def __rsub__(self, other):
        """
        Das subtrahieren einer Zahl weniger eines Bruches

        :raises: TypeError: Falscher
        Datentyp :param other: eine Zahl
        :return: Bruch
```

```
        """
        if type(other) is int:
            return Bruch((other*self.nenner)-self.zaehler,self.nenner)
        else:
            raise TypeError('Wrong
Datatype:'+type(other).__name__+'! Please use int')
```

Dies ist die Funktion rsub welche den welche einen Bruch von einer Zahl subtrahiert in reversierter Reihenfolge. Es folgt das Beispiel für die Verwendung von rsub.

```
b = Bruch(3,2)

a = 1 - b2
```

```
    def __mul__(self, other):
        """
        Das multiplizieren eines Bruches mit einem Bruch oder einer Zahl

        :raises: TypeError: Falscher Datentyp
        :param other: Ein Bruch oder eine Zahl
        :return: Bruch
        """
        if isinstance(other, Bruch):
            other = Bruch.__makeBruch(other)
            return Bruch(self.zaehler * other.zaehler, self.nenner
* other.nenner)
        elif type(other) is int:
            return Bruch(self.zaehler * other,
        self.nenner) else:
            raise TypeError('Wrong
Datatype:'+type(other).__name__+'! Please use int or Bruch')
```

Dies ist die Funktion mul welche den welche 2 Brüche multipliziert. Es folgt das Beispiel für die Verwendung von mul.

```
b  =  Bruch(3,2)
b2  =  Bruch(4,3)
m = b * b2
```

```
    def __imul__(self, other):
        """
        Das interne multiplizieren eines Bruches

        :param other: Bruch oder
        Zahl :return: Bruch
        """
        other = Bruch.__makeBruch(other)
        self = self*other
        return self
```

Dies ist die Funktion imul welche den welche 2 Brüche bzw einen Bruch und eine Zahl intern multipliziert. Es folgt das Beispiel für die Verwendung von imul.

```
b = Bruch(3,2)

b *= 1
b *= b
```

```
    def __rmul__(self, other):
        """
        Das multiplizieren einer Zahl mit einem Bruch

        :param other: eine
        Zahl :return: Bruch
        """
        return self.__mul__(other)
```

Dies ist die Funktion rmul welche den welche einen Bruch mit einer Zahl multipliziert in reversierter Reihenfolge. Es folgt das Beispiel für die Verwendung von rmul.

```
b = Bruch(3,2)

a = 1 * b2
```

```
    def __truediv__(self, other):
        """
        Das dividieren eines Bruches mit einem Bruch oder einer Zahl

        :raises: TypeError: Falscher Datentyp
        :param other: Ein Bruch oder eine Zahl
        :return: Bruch
        """
        if isinstance(other, Bruch):
            other = Bruch.__makeBruch(other)
            return Bruch(self.zaehler * other.nenner, self.nenner
* other.zaehler)
        elif type(other) is int:
            return Bruch(self.zaehler, self.nenner *
        other) else:
            raise TypeError('Wrong
Datatype:'+type(other).__name__+'! Please use int or Bruch')
```

Dies ist die Funktion truediv welche den welche 2 Brüche dividiert. Es folgt das Beispiel für die Verwendung von truediv.

```
b  =  Bruch(3,2)
b2 = Bruch(4,3)
m = b * b2
```

```
    def __itruediv__(self, other):
        """
        Das interne dividieren eines Bruches

        :param other: Bruch oder
        Zahl :return: Bruch
```

```
        """
        other = Bruch.__makeBruch(other)
        self = self / other
        return self
```

Dies ist die Funktion itruediv welche den welche 2 Brüche bzw einen Bruch und eine Zahl intern dividiert. Es folgt das Beispiel für die Verwendung von itruediv.

```
b = Bruch(3,2)

b /= 1
b /= b
```

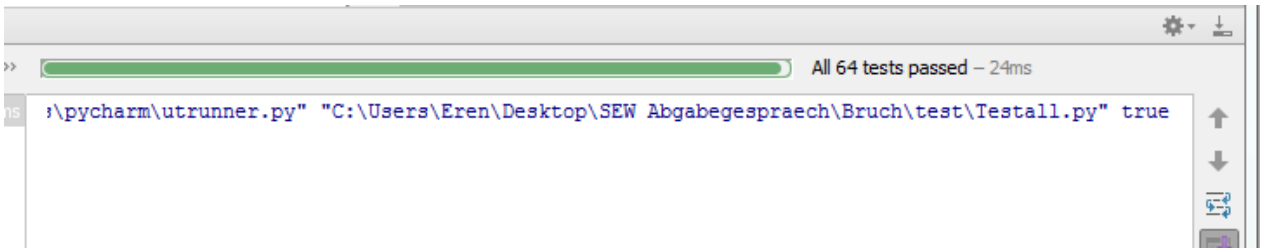
```
    def __rtruediv__(self, other):
        """
        Das dividieren einer Zahl durch einen Bruch

        :raises: TypeError: Falscher
        Datentyp :param other: eine Zahl
        :return: Bruch
        """
        if type(other) is int:
            return Bruch(other * self.nenner,1 * self.zaehler)
        else:
            raise TypeError('Wrong
Datatype:'+type(other).__name__+'! Please use int’)
```

Dies ist die Funktion rtruediv welche den welche einen Bruch durch eine Zahl dividiert in reversierter Reihenfolge. Es folgt das Beispiel für die Verwendung von rtruediv.

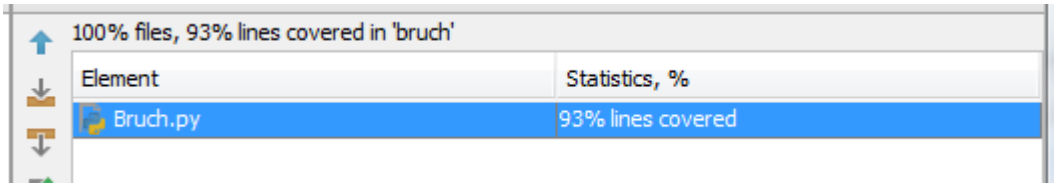
```
b = Bruch(3,2)

a = 1 / b2
```

Durchgeführte Tests

Alle vorgegebenen Tests wurden durchgeführt und bestanden.



Nach allen Tests ist die Coverage bei 93% wie der folgende Screenshot zeigt.



Grund des niedrigen Coverages

Die Coverage nach den Tests war bei ca 93% und deshalb nicht ausreichend für die Abgabe. Nach einer kurzen Untersuchung des Codes ist mir aufgefallen, dass bei den Vergleichs Funktionen nur getestet wird ob etwas True zurückgibt. Der Restliche Code welcher sich mit der Rückgabe eines False Wertes befasst war nicht getestet und ist deshalb auch nicht unter die Coverage gefallen.

Abgabeumfang

Die Abgabe der Aufgabe beinhaltet folgende Daten mit folgenden Strukturen:

- Bruchklasse als Bruch.py **(bruch)**
- Dieses Dokument als Protokoll.pdf **(protokoll)**
- Eine vollständige Sphinx Dokumentation der Klasse Bruch **(doc)**
- Coverage & Test Report als html **(html)**
- Testklasse **(test)**