Dokumentation von tools3.py und bruch.py

T. Hadamczik und T. Höfting

17.05.2019

Inhaltsverzeichnis

1 Schnittstellendokumentation tools3.py

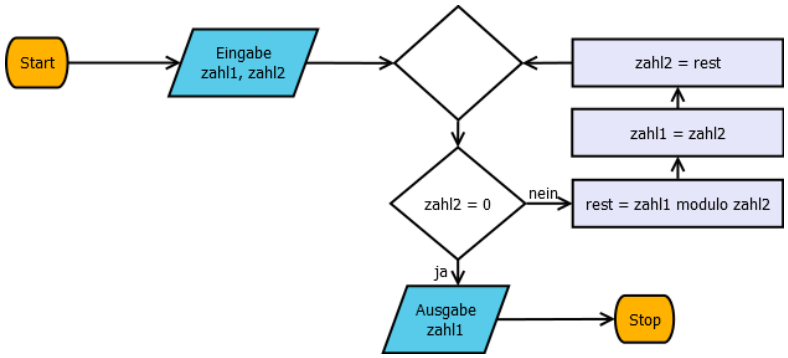
Die Datei stellt Funktionen zur Berechnung des ggT sowie des kgV von zwei Ganzzahlen zur Verfügung. Diese Funktionen werden später in `bruch.py` gebraucht.

1.1 Funktionen

1.1.1 `ggt()`

Diese Funktion berechnet den größten gemeinsamen Teiler von zwei ganzen Zahlen nach dem modernen euklidischen Algorithmus.
Input:
`zahl_a` *(int)* Die erste Zahl
`zahl_b` *(int)* Die zweite Zahl
Returns: *int* Den größten gemeinsamen Teiler der beiden eingegebenen Zahlen

Das Programm funktioniert nach dem folgenen Schema:



1.1.2 `kgv()`

Diese Funktion errechnet das kleinste gemeinsame Vielfache von zwei eingegebenen ganzen Zahlen.
Input:
`zahl_a` *(int)* Die erste Zahl
`zahl_b` *(int)* Die zweite Zahl
Returns: *int* Das kleinste gemeinsame Vielfache der beiden eingegebenen Zahlen

1.2 Hauptprogramm

In `tools3.py` ist eine `main()` Funktion implementiert. Sie dient zur Demonstration der beiden Funktionen `ggt()` und `kgv()`. Im Standardeingabegerät werden dem Programm zwei ganze Zahlen übergeben. Dieses berechnet zunächst den größten gemeinsamen Teiler der beiden Zahlen mit der im folgenden Abschnitt beschriebenen Methode und gibt diesen mit der `print()`-Funktion aus. Dann berechnet das Hauptprogramm das kleinste gemeinsame Vielfache von den beiden Zahlen und gibt diesen ebenfalls aus. Ein Beispieldurchlauf könnte wie folgt aussehen:
Auf der Konsole erscheint

```
Gebe die erste natürliche Zahl ein:
```

Eine beispielhafte Eingabe wäre "7". Dann erscheint

```
Gebe die zweite natürliche Zahl ein:
```

Eine beispielhafte Eingabe wäre "14". Im Hauptprogramm passiert Folgendes:

```
def main():
    """
    testet die beiden Funktionen von tools3.py
    """
    a = int(input("Gebe die erste natürliche Zahl ein: "))
    b = int(input("Gebe die zweite natürliche Zahl ein: "))
    print("Der ggT von "+ str(a)+ " und "+ str(b)+ " ist "+ str(ggt(a,b)))
    print("Das kgV von "+ str(a)+ " und "+ str(b)+ " ist "+ str(kgv(a,b)))
```

Die Standardausgabe liefert folgende Ausgabe:

```
Der ggT von 7 und 14 ist 7
Das kgV von 7 und 14 ist 14
```

2 Schnittstellendokumentation bruch.py

Die Funktionen `ggt()` und `kgv()` aus tools3.py werden importiert und in den Klassenmethoden benutzt. `bruch.py` enthält die namensgebende Klasse `Bruch`, welche Klassenmethoden enthält, um Operationen mit Brüchen auszuführen.

2.1 Die Klasse Bruch

Die Klasse dient dazu, einen Bruch abstrakt in seinen einzelnen Komponenten zu speichern und somit Operationen auf rationalen Zahlen durchzuführen, wie zum Beispiel die Addition.

2.1.1 Attribute

Bermerkung Alle Attribute der Klasse *Bruch* sind nicht-statisch.
`zaehler` *(int)* Der Betrag des Zählers des dargestellten Bruchs.
`nenner` *(int)* der Betrag des Nenners des dargestellten Bruchs.
`vorzeichen` *(bool)* Das Vorzeichen des Bruchs
False = Der Bruch ist positiv.
True = Der Bruch ist negativ.

2.1.2 Konstruktor

Im Konstruktor von der Klasse *Bruch* werden die Attribute festgelegt. Das Attribut `vorzeichen` wird auf `True` gesetzt, falls die beiden übergebenen Werte unterschiedliche Vorzeichen haben, sonst wird es auf `False` gesetzt. Die Attribute `zaehler` und `nenner` werden durch Anwendung der Funktion `abs()` auf die Beträge der entsprechenden Inputparameter gesetzt.
Input:
`zaehler` *(int)* Der Zähler des zu erzeugenden Bruchs.
`nenner` *(int)* Der Nenner des zu erzeugenden Bruchs.
Raises:
`ZeroDivisionError` *(Exeption)* Wird geworfen, falls der übergebene Nenner '0' ist.

2.1.3 Methoden

2.1.3.1 `kuerzen()`

Kürzt einen Bruch aus der Klasse Bruch. Hierzu wird `ggt()`aus `tools3.py` benutzt.
Input: - **Return:** *None*

2.1.3.2 `summe()`

Addiert ein Objekt der Klasse Bruch zu einem Objekt der gleichen Klasse, auf welches die Methode angewendet wird.
Input:
`bruch_b` *(Bruch)* Der Bruch, welcher hinzuaddiert werden soll.
Return: *Bruch* Die Summe des Bruches, auf welchen die Methode angewendet wird und des Eingabeparameters

2.1.3.3 `__add__()`

ist eine magic-Methode, die die Addition mit '+' sinnvoll für zwei Objekte der Klasse *Bruch* definiert. Die beiden Brüche werden addiert, wie wir es von rationalen Zahlen kennen.
Input:
`self` *(Bruch)* Der Bruch vor dem '+'.
`other` *(Bruch)* Der Bruch hinter dem '+'.
Return: *Bruch* Die Summe der Brüche vor und nach dem '+'.
Befehle folgender Form sind somit erklärt:

```
summe = Bruch(2,9) + Bruch(5,9)
```

2.1.3.4 `__str__()`

Hierbei handelt es sich ebenfalls um eine magic-Methode, die die Stringrepräsentation eines Objektes der Klasse sinnvoll festlegt.
Input: -
Returns: *string* eine Repräsentation des Objektes als String. Falls das Objekt eine ganze Zahl ist, wird es als solche ausgegeben. Andernfalls hat der String die Darstellung `zaehler/nenner`
Für `Bruch(-5,3)` wird z. B. Folgendes bei Anwendung von `print()` in der Standardausgabe ausgegeben:

```
-5/3
```

Bruch(3,1):

```
3
```

2.2 Hauptprogramm

Die `main()`-Funktion ist das Hauptprogramm. Bei Aufruf werden die Klasse und ihre Methoden zuerst durch einen vorgegebenen Datensatz getestet und anschließend kann der Benutzer selber zwei Brüche eingeben, deren Summe ausgegeben wird. Nach Aufruf der `main()`-Funktion erscheint Folgendes auf der Konsole:

```
Dies Summe von 3/7 und 4/9 ist = 55/63
Gebe Zaehler und Nenner des ersten Bruch durch ein Leerzeichen getrennt ein:
```

$\frac{3}{7}$ und $\frac{4}{9}$ ist der vorgegebene Datensatz, mit dem die Addition überprüft wird. Die Summe der beiden ist in der Tat $\frac{55}{63}$. Dann soll der Benutzer Zähler und Nenner des ersten Bruch durch ein Leerzeichen getrennt übergeben. Die Eingabe muss die Form

```
[<Nenner> <Zähler>]
```

haben. Wenn der eingegebene Nenner gleich 0 war ist die Eingabe fehlerhaft und auf der Konsole erscheint

```
Gebe den Nenner des ersten Bruchs ein, der ungleich 0 sein muss:
```

Nun muss der Benutzer so lange den Nenner korrigieren, bis er einen Wert ungleich 0 hat. Analog wiederholt sich dies für den zweiten Bruch:

```
bruch2 = input("Gebe Zaehler und Nenner \
des zweiten Bruchs durch ein Leerzeichen getrennt ein:").split()
bruch2 = (int(bruch2[0]), int(bruch2[1]))
while bruch2[1] == 0:
    neuer_nenner2 = int(input("Gebe den Nenner des zweiten Bruchs ein, \
der ungleich 0 sein muss: "))
    bruch2 = (bruch2[0], neuer_nenner2)
```

Dann werden die Brüche wie folgt addiert und an die Standardausgabe gegeben:

```
bruch_summe = Bruch(bruch1[0], bruch1[1]) + Bruch(bruch2[0], bruch2[1])
print("Die Summe der beiden Brüche ist " + str(bruch_summe) + ".")
```

In der Standardausgabe erscheint, falls z. B. '[-1 2]' und '[1 3]' eingegeben wurden:

```
- 1/6
```