Programmierkurs Python I – WS 10/11

Übung 5

---

1 Rationale Zahlen (3 + 2 Punkte)

Definiere eine Klasse, die Bruchzahlen darstellt. Der Konstruktor soll Zähler und Nenner als Argumente nehmen. Die Bruchzahl-Objekte sollen mit den Standard-Python-Operatoren `+`,`-`,`*` und `/` funktionieren:

```
>>> Rat(2,5) + Rat(3,4)
23/20
```

Um das zu erreichen, muss die Klasse die Methoden `__add__(self,b)`, `__sub__(self,b)`, `__mul__(self,b)` und `__truediv__(self,b)` implementieren, die ihrerseits wieder `Rat`-Objekte zurückgeben. Für die String-Ausgabe der Objekte (Beispiel) müsst ihr die Methode `__repr__(self)` implementieren, die einen String zurück gibt.

Bonusaufgabe (2 P.): Implementiere den Euklidischen Algorithmus zum Errechnen des größten gemeinsamen Teilers und benutze ihn für eine Methode, die die `Rat`-Objekte kürzt. (Referenz: Foliensatz 2, Folie 32 ff.) Wie im Beispiel oben sollen bei allen Rechenoperationen gekürzte Brüche zurückgegeben werden.

2 Einkaufsliste (4 + 2 Punkte)

Implementiere eine Klasse `ShoppingList`, die Einkaufsliste bzw. Rechnung simuliert. Die Klasse soll drei Methoden unterstützen: `add(self,item)` fügt der Liste ein Element hinzu, `calculateSum(self)` berechnet den aktuellen Gesamtpreis und `getList(self)` gibt einen String zurück, auf dem alle Produkte mit Menge und Preis notiert sind. Geh davon aus, das jede Art von Item nur einmal zur Liste hinzugefügt wird.

Für die Produkte soll eine Klasse `Item` implementiert werden. Ein Name (String) und ein Grundpreis (float), werden in der Init-Methode (`__init__(self,name,price)`) übergeben und gespeichert. Beides soll jeweils mit `getName(self)` und `getPrice(self)` wieder zurück gegeben werden.

Von der Item-Klasse soll die Klasse `UncountableItem` abgeleitet werden: hier gibt es nur einen Preis pro Kilo. Im Konstruktor soll man optional eine Menge des

Items mitgeben können (`__init__(self, name, price, quantity=...)`). Außerdem soll `getPrice(self)` den Preis der angegebenen Menge zurückgeben.

Auf der gedruckten Einkaufsliste soll man normale `Item`s von `UncountableItem`s unterscheiden können, indem bei der Mengenangabe entweder „kg“ oder „stck“ steht. Denke dir hierfür eine sinnvolle Lösung aus. *Bonusaufgabe 1:* Wenn zur Liste ein `Item` hinzugefügt wird, von dessen Sorte schon eines vorhanden war, soll entweder die Stückzahl um eins erhöht oder, für `UncountableItem`, das Gewicht entsprechend erhöht werden (siehe Beispiel unten).

Bonusaufgabe 2: Implementiere die Einkaufslisten so, dass man mit „+“ zwei von ihnen zu einer großen addieren kann. Gleiche Produkte sollen zusammengefasst werden.

```
>>> liste = ShoppingList()
>>> apfel = UncountableItem("Apfel",1.99,0.7)
>>> kekse = Item("Kekse", 2.05)
>>> liste.add(apfel)
>>> liste.add(kekse)
>>> liste.add(apfel) #Nur möglich bei Bonusaufgabe 1, sonst ignorieren.
>>> print(liste.getList())
Apfel 1.4kg  2.79  #Hinweis: falls hier nicht gerundet wird, ist das ok.
Kekse 1stck  2.05
Summe        4.84
```

3 Ganze & Rationale Zahlen (Bonus, 2 Punkte)

Implementiere eine Klasse `Int` für ganze Zahlen, so dass man ganze Zahlen mit Bruchzahlen addieren, subtrahieren, etc. kann (natürlich nur Instanzen der `Int`-Klasse).

```
>>> r = Rat(1,2)
>>> i = Int(3)
>>> r + i
Rat(7,2)
```

Hinweis: eine einfache Lösung besteht darin, ganze Zahlen von Bruchzahlen abzuleiten. Ggf. muss die Implementierung der Klasse für Bruchzahlen leicht geändert werden.

4 Default-Dictionaries (Bonus, 3 Punkte)

In der Vorlesung wurde die Klasse `Defaultdict` vorgestellt. Die Implementierung hat eine wichtige Einschränkung: Sie funktioniert nur dann wirklich sinnvoll, wenn der Default-Wert ein unveränderlicher Typ (z.B. eine Zahl) ist. Wie müsste man die Implementierung anpassen, um z.B. Dictionaries als Default-Wert zuzulassen?

```
>>> d = Defaultdict(...)
>>> d[47][11] = 23
>>> d[47][11]
23
>>> d[17][4] = 99
>>> d[17][4]
99
>>> d[47][11]
23
```

Versuche, den Lösungsansatz möglichst allgemein zu halten.

Abgabe bis Donnerstag, 25.11.10, 14:00 Uhr per Mail an `stth@coli.uni-sb.de` **und** `regneri@coli.uni-sb.de`