Vorwort

Alle Lösungen zur Aufgabe sind getestet und geschrieben auf einem 64-Bit GNU/Linux-Rechner mit Python 2.7.8 und 3.4.2. Die Dokumentation ist geschrieben in Markdown und später konvertiert zu einem PDF.

Alle Lösungen sind innerhalb von zwei Wochen während der Schulzeit entstanden.

Der Code und die Dokumentation werden mit Git verwaltet und werden nach Abgabetermin auf GitHub verfügbar sein.

Zahlenspiel

Die Aufgabe ließe sich mit `random` s `randrange()` lösen. Zu Beginn der Bruchsuche müssen `p` und `q` gefunden werden, die der Anforderung des Schwierigkeitsgrades entsprechen und sich, geschrieben als Bruch, nicht kürzen lassen. Das Nicht-Kürzen-Lassen lässt sich mit dem Greatest Common Divisor (größter gemeinsamer Teiler) berechnen - ist er eins, lässt sich `p/q` nicht kürzen. Um die Zahlen `p` `q` innerhalb der Schwierigkeitsgrenzen zu halten ist es sinnvoll, nach dem Generieren einer zufälligen Zahl `Maximum > p > 1` die Zahl `q` abhängig davon zu finden: `Maximum - p > q > Minimum - p`

TODO Minimum - p?

Als Nächstes gilt es, einen Bruch zu finden, der der Schwierigkeitsstufe entsprechend lang genug ist. Dazu sucht die Funktion, angefangen bei einem Wert von 2, nach einem `x` welches für eine zufriedenstellende Länge des Bruches sorgt. Schießt die Länge über das Ziel hinaus, wird ein neuer Bruch gesucht.

`bruch(Stufe)` gibt die Variablen `a, b, p, q` zurück, mit denen ein Bruch `a/b = p/q` der Schwierigkeitsstufe gebildet werden kann. Durch `aufgaben(Stufe, Anzahl)` können mehrere Aufgaben der gleichen Stufe mit `print` ausgegeben werden. Schwierigkeitsstufen gehen von 0 - leicht über 1 - mittel bis 2 - schwer.

Beispiele

```
>>> import zahlenspiel
>>> zahlenspiel.aufgaben(1, 5)
75 / 105 = 5 / 7
104 / 39 = 8 / 3
100 / 10 = 10 / 1
112 / 24 = 14 / 3
108 / 45 = 12 / 5
>>> zahlenspiel.aufgaben(2, 1)
49 / 105 = 7 / 15
>>> zahlenspiel.aufgaben(0, 10)
60 / 10 = 6 / 1
70 / 10 = 7 / 1
10 / 15 = 2 / 3
70 / 10 = 7 / 1
10 / 40 = 1 / 4
15 / 12 = 5 / 4
10 / 50 = 1 / 5
10 / 30 = 1 / 3
20 / 12 = 5 / 3
80 / 10 = 8 / 1
```

Pong

Der Pong-Bot ist als 'TPong' des Benutzers 'tmo' angemeldet. Der Bot berechnet die Flugbahn des Balls und versucht, diese mit einem Delta-X und Delta-Y verschiedener Messwerte hervorzusagen. Ist der Ball für eine Berechnung zu nahe, begibt sich der Schläger auf Höhe des Balls.

Aufgrund Zeitmangels ist der Code nicht fertig geworden und der Bot erfüllt seine Funktion nicht perfekt.

Mobile

Textuelle Darstellung

Die textuelle Darstellung setzt sich aus einer Liste von Position-Gewichts-Tuples zusammen. Ein Gewicht ist hierbei ebenfalls eine Liste. Positionen sind auf der einen Seite negativ, auf der anderen positiv. Ein Balken lässt sich daran erkennen, dass er mehr als ein Gewicht hat. Eine Figur ist an der Position 0 mit dem Gewicht der entsprechenden Figur. Beispiel für ein Mobile mit einer Figur des Gewichts 2: `[(0, 2)]` Beispiel für einen Balken mit zwei Figuren des Gewichts 3: `[(-3, 3), (3, 3)]`

Das abgebildete Mobile sieht in der textuellen Darstellung wie folgt aus:

```
[(-4, [(-2, [(0, 1)]), (1, [(0, 2)])]), (-1, [(0, 2)]), (2, [(0, 3)]
```

Umsetzung

Die Lösung besteht aus zwei Funktionen, `split(alle_Figuren_an_einem_Balken)`, welche eine textuelle Darstellung verteilt. Diese eingehende Darstellung ist ein einziger Balken mit allen Figuren, die mithilfe dieser Funktion ausgeglichen werden. Im Endergebnis hat dann jeder Balken maximal vier Gewichte. Die Liste, die zurückgegeben wird, ähnelt der textuellen Darstellung, jedoch ist statt der Position das Gewicht des Gewichts in dem Tuple enthalten. Diese Liste wird von der zweiten Funktion `weight(Liste_erstellt_durch_split)` angeglichen. Aus Benutzerfreundlichkeitsgründen fasst eine dritte Funktion `mobile(List)` diese zwei zusammen. Sie benötigt nur noch eine Liste aller Gewichte für das Mobile.

Die Funktion `split` sortiert alle Items der mitgelieferten Liste in vier Plätze ein (realisiert mit dem Modulo-Operator). Das Gesamtgewicht dieser vier Plätze wird berchnet, dann wird jedes Gewicht in diesen vier Plätzen durch einen rekursiven Funktionsaufruf erneut verteilt.

`weight` berechnet die Position mit der Gegenzahl des Gesamtewichts des Gewichts an der gegenüberliegenden Position. Ist die Anzahl an Gewichten auf dem Balken ungerade, wird ein Gewichte in die Mitte (Position 0) platziert. Durch diese Verteilung gleichen sich die jeweils äußersten und innersten Gewichtpaare aus. Auch diese Funktion ruft sich selbst auf, um die Gewichte von der Spitze des Mobiles hinab bis zu den Figuren alle zu verteilen.

Beispiele

```
>>> import mobile
>>> mobile.mobile([1, 4])
[(-4, 1), (1, 4)]
>>> mobile.mobile([4, 6, 2, 3])
[(-3, 4), (-2, 6), (6, 2), (4, 3)]
>>> mobile.mobile([1, 3, 5, 2, 3, 6])
[(-2, [(-3, 1), (1, 3)]), (-5, [(-6, 3), (3, 6)]), (9, [(0, 5)]), (4
>>> mobile.mobile([4, 6, 2, 3, 45, 6, 3, 6, 1, 8, 5, 3, 67, 24])
[(-12, [(-67, 4), (-1, 45), (45, 1), (4, 67)]), (-10, [(-24, 6), (-8
```

{"message":"API rate limit exceeded for 79.193.24.98. (But here's the good news: Authenticated requests get a higher rate limit. Check out the documentation for more details.)","documentation_url":"https://developer.github.com/v3/#rate-limiting"}

{"message":"API rate limit exceeded for 79.193.24.98. (But here's the good news: Authenticated requests get a higher rate limit. Check out the documentation for more details.)","documentation_url":"https://developer.github.com/v3/#rate-limiting"}

{"message":"API rate limit exceeded for 79.193.24.98. (But here's the good news: Authenticated requests get a higher rate limit. Check out the documentation for more details.)","documentation_url":"https://developer.github.com/v3/#rate-limiting"}

{"message":"API rate limit exceeded for 79.193.24.98. (But here's the good news: Authenticated requests get a higher rate limit. Check out the documentation for more details.)","documentation_url":"https://developer.github.com/v3/#rate-limiting"}