Реализация класса Fraction для работы с рациональными числами

Создайте новый класс `Fraction` для работы с рациональными числами. Реализуйте арифметические и логические операции с рациональными числами. В задачах `A-G, J-M` вам требуется сдать программу, содержащую **ТОЛЬКО** описание класса `Fraction` с необходимыми методами. В этих задачах ваша программа не должна ничего считывать и выводить.

Обратите внимание, что все бинарные операции (арифметические и логические) должны корректно выполняться в случае, если один из операндов — целое число.

Документация, где можно прочитать про стандартные методы, а также перегрузку (*overloading*) арифметических и логических операций можно здесь.

A. Реализация методов `__init__` и `__str__`.

Метод `__init__` может принимать на вход пару целых чисел, второе из которых не равно нулю, одно число или вообще не принимать никаких аргументов и определять атрибуты экземпляра класса, отвечающие за числитель и знаменатель дроби. Числитель — целое число, знаменатель — натуральное число, при этом дробь должна быть несократима.

- 2 аргумента — числитель и знаменатель;
- 1 аргумент — числитель, знаменатель равен 1;
- без аргументов — числитель равен 0, знаменатель равен 1.

Метод `__str__` должен возвращать строковое представление дроби в виде, показанном в тестах.

Задачи `B-F` проверяются также на всех тестах из предыдущих задач.

Input	Output

`print(Fraction(2, 4))`
`print(Fraction(30, -90))`
`print(Fraction(8, 2))`
	`1/2`
	`-1/3`
	`4`

B. Реализация арифметических операций — сложение и унарный плюс (`__add__`, `__radd__`, `__pos__`)

Рациональные числа должны поддерживать сложения друг с другом и с целыми числами.

Input	Output

`a = Fraction(2, 4)`
`b = Fraction(30, -90)`
`c = Fraction(3, 7)`
`print(a + b)`
`print(a + b + c + 1)`
`c += (+a + b)`
`print(c)`
`print(1 + a)`
	`1/6`
	`67/42`
	`25/42`
	`3/2`

C. Реализация арифметических операций — вычитание и унарный минус (`__sub__`, `__rsub__`, `__neg__`)

Input	Output

`a = Fraction(2, 4)`
`b = Fraction(30, -90)`
`c = Fraction(3, 7)`
`print(a - b)`
`print(-a - b - c - 2)`
`a -= b + c`
`print(a)`
`print(1 - c)`
	`5/6`
	`-109/42`
	`17/42`
	`4/7`

D. Реализация арифметических операций — умножение (`__mul__`, `__rmul__`)

Input	Output

`a = Fraction(1, 2)`
`b = Fraction(3, -7)`
`c = Fraction(1, 5)`
`print(a * b)`
`c *= a + b`
`print(c)`
`print(3 * a)`
	`-3/14`
	`1/70`
	`3/2`

E. Реализация арифметических операций — деление (результат — дробь, `__truediv__`, `__rtruediv__`)

Input	Output

`a = Fraction(1, 2)`
`b = Fraction(3, -7)`
`c = Fraction(1, 5)`
`print(a / Fraction(2, 3))`
`b /= c / a`
`print(b)`
`print(4 / Fraction(3, -7))`
	`3/4`
	`-15/14`
	`-28/3`

F. Реализация логических операций — сравнения
(==, !=, <, <=, >, >=, соответственно (__eq__, __ne__, __lt__, __le__, __gt__, __ge__)

Input	Output
a = Fraction(2, 5) b = Fraction(5, 12) c = Fraction(3, 7) d = Fraction(1, 35) print(a > b) print(a != c) print(a + d == c) print(b >= c) print(c < a) print(c <= a + b) print(1 < a)	False True True False False True False

G. *Ближайшая аликвотная дробь*

По данной положительной дроби, не превосходящей 1, найдите максимальную дробь с числителем, равным 1, не превосходящей данную.

Для решения задачи реализуйте метод max_aliquote.

На вход программе подаётся строка вида A/B, где A и B — натуральные числа, десятичная запись которых содержит не более 100 знаков.

Требуется вывести дробь с числителем, равным 1, такую, что $\frac{1}{C}$ — максимальная дробь с числителем, равным 1, такая что $\frac{1}{C} \leqslant \frac{A}{B}$.

Input	Output
print(Fraction(2, 5).max_aliquote())	1/3
print(Fraction(1, 4).max_aliquote())	1/4
print(Fraction(23, 101).max_aliquote())	1/5
print(Fraction(45, 46).max_aliquote())	1/2

H. *Египетские дроби*

Разложением дроби на сумму египетских дробей называется представление данной дроби в виде суммы дробей с числителем равным 1 и разными знаменателями (дроби с числителем равным 1 ещё называют *аликвотными* дробями).

Существует несколько алгоритмов разложения данной дроби на сумму египетских дробей. Намёк на один из них — предыдущая задача.

Дана дробь, не превосходящая 1. Найти разложение данной дроби на сумму египетских дробей и вывести это разложение (см. примеры). Если существует несколько разложений, вывести любое.

На вход программе подаётся строка в формате, описанном в задаче G.

Программа должна вывести выражение — сумму различных аликвотных дробей или одну дробь, если заданная дробь является аликвотной. Значение выведенного выражения должна быть равно данной дроби.

Input	Output
2/5	1/3+1/15
3/7	1/3+1/11+1/231
1/13	1/13
5/121	1/25+1/757+1/763309+1/873960180913+1/1527612795642093418846225

Определение: непрерывная (или цепная) дробь (*continued fraction*) — это конечное или бесконечное выражение вида:

$$[a_0; a_1, a_2, \ldots] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}}$$

где a_0 — целое число, остальные a_i — натуральные числа, причём если дробь конечная, то последнее значение отлично от единицы. Число представляется конечной цепной дробью тогда и только тогда, когда оно рационально.

Сейчас мы добавим нашим дробям возможность превращения в цепную или периодическую десятичную дробь и обратно. Начнём с конструирования рационального числа из цепной дроби.

Добавим метод `from_continued` в класс `Fraction`, который по массиву или кортежу целых чисел — элементов цепной дроби — будет создавать дробь. Работать это должно так:

```
class Fraction:
    ...
    def from_continued(terms):
        ...
        return Fraction(p, q)

x = Fraction.from_continued([1, 1, 2]) # 5/3
y = Fraction.from_continued([0, 2, 4, 8, 16]) # 532/1193
```

Обратите внимание, здесь внутри метода `from_continued` нам не нужен аргумент `self` — экземпляр класса. При вызове как в примере выше он и не будет передан. Но этот метод можно вызвать и так:

```
>>> x = Fraction(1, 2)
>>> y = x.from_continued([1, 1, 2])
TypeError: from_continued() takes 1 positional argument but 2 were given
```

Нужно дать знать интрепретатору `python`, что экземпляр класса нам никогда не потребуется. Для этого перед определением метода нужно добавить декоратор `@staticmethod`. Вот так:

```
class Fraction:
    ...
    @staticmethod
    def from_continued(terms):
        ...
        return Fraction(p, q)
```

I. *Получение значения дроби по её разложению в цепную дробь*

Добавьте метод `from_continued` в класс `Fraction`, который по массиву или кортежу целых чисел — элементов цепной дроби — будет создавать дробь.

Input	Output
`print(Fraction.from_continued([1, 1, 2]))`	`5/3`

`print(Fraction.from_continued([3]))`
`print(Fraction.from_continued([1, 2, 3]))`
`print(Fraction.from_continued([0, 2, 4, 8, 16]))`
`print(Fraction(1, 3) + Fraction.from_continued([1, 1, 2]))`
	`3`
	`10/7`
	`532/1193`
	`2`

J. *Разложение данной дроби в цепную дробь*

Добавьте метод `to_continued` в класс `Fraction`, который возвращает массив целых чисел, представляющий собой разложение данной дроби в цепную дробь.

Input	Output
`print(Fraction(5, 3).to_continued())`	`[1, 1, 2]`
`print(Fraction(3).to_continued())` `print(Fraction(10, 7).to_continued())`	`[3]` `[1, 2, 3]`
`print(Fraction(532, 1193).to_continued())` `print(Fraction(-7, 3).to_continued())`	`[0, 2, 4, 8, 16]` `[-3, 1, 2]`

K. *Ближайшая дробь*

По данной дроби $\frac{A}{B}$ ($0 < A, B < 10^{10}$) найти ближайшую к ней дробь со знаменателем, не превышающим данное число Q ($Q < 20000$), где $Q < B$. Если для данной дроби существует несколько дробей, равноудалённых от неё — выведите дробь с наименьшим знаменателем. Если существует несколько дробей с наименьшими равными знаменателями, равноудалённых от данной, выведите наименьшую дробь.

Реализуйте эту функцию в виде метода `limit_denominator`, принимающего в качестве параметра число Q.

Input	Output
`print(Fraction(10, 23).limit_denominator(22))`	`7/16`

L. *Получение по периодическому представлению дроби её значения*

Дана строка, содержащая десятичную запись дроби. Если строка содержит одну точку, то слева от неё — запись целого числа. Справа от точки последовательность цифр от 0 до 9, при этом возможно какой-то непустой суффикс этой последовательности взят в круглые скобки.

Суммарная длина строки, в которой записано периодическое представление не превышает 10^5 символов.

Требуется вычислить и вывести значение соответствующей несократимой дроби.

Реализуйте это в виде метода `from_repeating` (по-английски периодическая дробь — *repeating decimal*).

Input	Output
`print(Fraction.from_repeating("1.41(6)"))`	`17/12`

`print(Fraction.from_repeating("0.(0588235294117647)"))`
`print(Fraction.from_repeating("0.(629)"))`
`print(Fraction.from_repeating("0.25"))`
`print(Fraction.from_repeating("-2"))`
	`1/17`
	`17/27`
	`1/4`
	`-2`

M. *Получение по значению дроби её периодического представления*

Дана дробь A/B ($A, B < 10^7$). Требуется вывести её представление в виде конечной десятичной или бесконечной периодической дроби.

Обратите внимание, что дроби допускают различные способы записи. Проверьте, что соблюдены следующие правила:

- Конечная десятичная дробь не содержит незначащих нулей.
$$\frac{1}{8} = 0.12500 = 0.125$$
- Бесконечная периодическая дробь записана без предпериода, если это возможно. Например,
$$\frac{1231}{9999} = 0.123(1123) = 0.(1231)$$
- Если предпериод есть, его длина минимальна:
$$\frac{211}{990} = 0.21(31) = 0.2(13)$$
- Длина периода наименьшая из возможных: $\frac{1}{3} = 0.(33) = 0.(3)$

Реализуйте эту функцию в виде метода `to_repeating`.

Указание: надо научиться вычислять длину предпериода, и далее вычислять цифры периода. При этом дополнительная память, которую вы используете не должна зависеть от длины периода.

Чтение по теме: Квант, 2000г. “Периодические дроби”

Input	Output
`print(Fraction(17, 12).to_repeating())`	`1.41(6)`

`print(Fraction(1, 17).to_repeating())`
`print(Fraction(17, 27).to_repeating())`
`print(Fraction(1, 4).to_repeating())`
	`0.(0588235294117647)`
	`0.(629)`
	`0.25`

N** *Ближайшая дробь (быстрый алгоритм)*

По данной дроби $\frac{A}{B}$ ($A, B < 10^{100}$) найти ближайшую к ней дробь со знаменателем, не превышающим данное число Q, где $Q < B$. Если для данной дроби существует несколько дробей, равноудалённых от неё — выведите дробь с наименьшим знаменателем.

Эта задача в точности совпадает с задачей K, отличаясь только ограничениями на величину числителя и знаменателя данной дроби.

Литература по теме:

- В.В. Прасолов “Задачи по алгебре, арифметике и анализу”
- И.Н. Сергеев, С.Н. Олехник, С.Б. Гашков “Примени математику”
- Г. Дэвенпорт “Высшая арифметика”
- А.Я. Хинчин “Цепные дроби”
- В.И. Арнольд “Цепные дроби”
- Wikipedia

Input	Output
`print(Fraction(10, 23).limit_denominator(22))`	`7/16`

O. *Игра с калькулятором*

У вас есть калькулятор, который умеет выполнять арифетические операции (сложение, вычитание, умножение и деление). Результат деления целых чисел (в случае, когда он не является целым числом) калькулятор умеет показывать вам с точностью до, скажем, первых 50 десятичных разрядов.

Представим себе следующую игру. Вы можете выбрать один из возможных вариантов действий:

- выбираете два случайных числа от 1 до 100000 и делите одно на другое. Получаете ответ в виде дроби, и выписываете первые 50 десятичных разрядов его дробной части.
- водите пальцем по кнопкам калькулятора и получаете *случайную* последовательность из 50 цифр.

Ваша задача: написать программу, которая в данных условиях сумеет определить по последовательности из 50 цифр, какой из способов был использован для её получения. Если число получено при помощи деления, выведите `RATIONAL`, иначе выведите `RANDOM`.

Тесты в этой задаче закрытые. В каждом тесте в первой строке указано натуральное число N, после чего N строк, в каждой из которых записана последовательность из 50 цифр без разделителей.

Литература по теме приведена в предыдущей задаче, а кроме того можно почитать, например, замечательную книжку «Математический дивертисмент» Сергея Табачникова и Дмитрия Фукса.

Input	Output
81504858796678001868515099327620295960919576801285	RANDOM
04098472842529894827834605964558420976804495029534	RATIONAL
65620886155843687447027212233783564459993147350008	RATIONAL
79295424616572769488202619879798409853758032416804	RATIONAL
71368110823794816367183940729755459969648507089869	RATIONAL
07230935482661632464220914285636669261174201759061	RANDOM
43310946362920969724658715119924460828397453981816	RANDOM
90649125634120798608004383559140501051799143861756	RANDOM
79485406521331383752787840945353627149102220745097	RANDOM
05471764553143675247574915667388590104261979378050	RANDOM
24717702242571855712176717634740996700813924196593	RANDOM
21312930451324098918111530306951235568388698964419	RATIONAL
57883145363408521303258145363408521303258145363408	RATIONAL
01343024319940693740495976222241919349472954757105	RATIONAL
48801448623934179631989798866771907139267742835221	RANDOM
74348181090299672014058440458666709321435862483290	RANDOM
67249820645206154629459022682671783225975622784159	RANDOM
64675440512886031049490848581904652357555031463319	RANDOM
10787289277703996105308724182035123346197096394835	RATIONAL
09287249830099443835730433697174796466068431783193	RATIONAL

Список методов, использующихся для перегрузки стандартных операций:

Операторы сравнения

Метод	Использование
`__lt__(self, other)`	`self < other`
`__le__(self, other)`	`self <= other`
`__eq__(self, other)`	`self == other`
`__ne__(self, other)`	`self != other`
`__gt__(self, other)`	`self > other`
`__ge__(self, other)`	`self >= other`

Сложение

Метод	Использование
`__add__(self, other)`	`self + other`
`__radd__(self, other)`	`other + self`
`__iadd__(self, other)`	`self += other`

Вычитание

Метод	Использование
`__sub__(self, other)`	`self - other`
`__rsub__(self, other)`	`other - self`
`__isub__(self, other)`	`self -= other`

Умножение

Метод	Использование
`__mul__(self, other)`	`self * other`
`__rmul__(self, other)`	`other * self`
`__imul__(self, other)`	`self *= other`

Деление

Метод	Использование
`__truediv__(self, other)`	`self / other`
`__rtruediv__(self, other)`	`other / self`
`__itruediv__(self, other)`	`self /= other`

Целочисленное деление

Метод	Использование
`__floordiv__(self, other)`	`self // other`
`__rfloordiv__(self, other)`	`other // self`
`__ifloordiv__(self, other)`	`self //= other`
`__divmod__(self, other)`	`divmod(self, other)`

Остаток

Метод	Использование
`__mod__(self, other)`	`self % other`
`__rmod__(self, other)`	`other % self`
`__imod__(self, other)`	`self %= other`

Возведение в степень

Метод	Использование
`__pow__(self, other)`	`self ** other`
`__rpow__(self, other)`	`other ** self`
`__ipow__(self, other)`	`self **= other`

Отрицание, модуль

Метод	Использование
`__pos__(self)`	`+self`
`__neg__(self)`	`-self`
`__abs__(self)`	`abs(self)`

Преобразования к стандартным типам

Метод	Использование
`__bool__(self)`	`bool(self), if self`
`__int__(self)`	`int(self)`
`__float__(self)`	`float(self)`
`__str__(self)`	`str(self)`
`__repr__(self)`	`repr(self)`
`__round__(self, digits = 0))`	`round(self, digits)`