

Реализация класса Poly (многочлены)

В этом листке требуется реализовать класс многочленов Poly, используемый для представления многочленов и операций над ними. Как и в листке по реализации класса Дроби, при сдаче в тестирующую программу необходимо добавить после описания класса строку `exec(open('input.txt').read())`.

Преобразования к стандартным типам

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

A. Конструктор и repr

Создайте новый класс Poly с единственным атрибутом: `_coeff` — списком коэффициентов многочлена. Конструктор `__init__` на данном этапе может принимать только кортеж чисел типов `int` или `float`.

Метод `__repr__` должен возвращать строку, из которой можно создать в точности такой же многочлен.

Добавьте также константу `X` — многочлен `Poly((0, 1))`. Переменная с таким именем должна быть создана после описания класса, до вызова `exec`.

Input	Output
<code>print(repr(Poly((1, 2, 3))))</code>	<code>Poly((1, 2, 3))</code>
<code>print(repr(Poly((1, 2.34, 4.56, 7))))</code>	<code>Poly((1, 2.34, 4.56, 7))</code>
<code>print(repr(X))</code>	<code>Poly((0, 1))</code>

B. Конструктор — 2

Доработайте конструктор `__init__` так, чтобы он мог принимать следующие типы аргументов:

- Без аргументов `Poly()`. В таком случае создается многочлен степени 0 с единственным коэффициентом 0.
- Один аргумент типа `int` или `float`. В этом случае создается многочлен степени 0 с одним коэффициентом;
- Один аргумент — итерируемый объект, возвращающий элементы типа `int` или `float` (например, список, кортеж, `range`, `map` и т.п.). При этом нулевые старшие коэффициенты обрезаются.

Проверить, что объект является итерируемым, можно при помощи `hasattr(объект, '__iter__')`.

Input	Output
<code>print(repr(Poly()))</code>	<code>Poly((0))</code>
<code>print(repr(Poly(1)))</code>	<code>Poly((1))</code>
<code>print(repr(Poly(1.23)))</code>	<code>Poly((1.23))</code>
<code>print(repr(Poly((1, 2.34, 4.56, 7, 0))))</code>	<code>Poly((1, 2.34, 4.56, 7))</code>
<code>print(repr(Poly([1, 2.34, 4.56, 7, 0, 0, 0])))</code>	<code>Poly((1, 2.34, 4.56, 7))</code>
<code>print(repr(Poly(range(10))))</code>	<code>Poly((0, 1, 2, 3, 4, 5, 6, 7, 8, 9))</code>
<code>print(repr(Poly(map(lambda x: x ** 4, range(5)))))</code>	<code>Poly((0, 1, 16, 81, 256))</code>
<code>print(repr(Poly(map(int, '1 2 3'.split()))))</code>	<code>Poly((1, 2, 3))</code>

C. Конструктор — 3

Доработайте конструктор `__init__` так, чтобы он мог принимать следующие типы аргументов:

- Один аргумент типа `int` или `float`. В этом случае создается многочлен степени 0 с одним коэффициентом;
- Строка, содержащая коэффициенты многочлена типа `int` или `float` (от младшего к старшему) через пробел;
- Многочлен (объект класса Poly). Конструктор должен создавать копию списка коэффициентов.
- Один аргумент — итерируемый объект, возвращающий элементы типа `int` или `float` (например, список, кортеж, `range`, `map` и т.п.). При этом нулевые старшие коэффициенты обрезаются.

Проверить, что объект является итерируемым, можно при помощи `hasattr(объект, '__iter__')`.

- Без аргументов `Poly()`. В таком случае создается многочлен степени 0 с единственным коэффициентом 0.

Input	Output
<code>print(repr(Poly('1')))</code>	<code>Poly((1))</code>
<code>print(repr(Poly('1 2 3')))</code>	<code>Poly((1, 2, 3))</code>
<code>print(repr(Poly('1 2.34 5.67 0')))</code>	<code>Poly((1, 2.34, 5.67))</code>
<code>A = Poly('1 2.34 5.67 0')</code>	<code>False</code>
<code>B = Poly(A)</code>	<code>False</code>
<code>print(A is B)</code>	<code>True</code>
<code>print(A._coeff is B._coeff)</code>	
<code>print(A._coeff == B._coeff)</code>	

D. `__str__`

Реализуйте метод `__str__`, который будет выводить удобное для человека представление многочлена:

- Одночлены в многочлене записываются от старшего к младшему с переменной x через знаки $+$ и $-$.
- Одночлены, коэффициент при которых равен 0, пропускаются.
- Между коэффициентом и переменной знаков нет, например: $2x$.
- Степень многочлена записывается в верхнем регистре, соответствующими символами *Unicode* (см. таблицу ниже), например, x^{10} .
- Если коэффициент отрицательный, то вместо знака $+$ перед слагаемым пишется $-$.
- Пробелы ставятся только вокруг знаков $+$ и $-$, но если коэффициент перед старшим членом отрицательный, то между знаком и модулем коэффициента пробела нет.
- Свободный член записывается без x^0 , вместо x^1 записывается x .
- Коэффициенты, равные по модулю 1, не пишутся, например: $-x^2$. Исключением является свободный член, он пишется и в том случае, когда равен 1.
- Если коэффициент имеет тип `float`, то округляется до 3 знаков после запятой, например: $-0.333x$. Про правила округления в языке Python можно почитать здесь [\[ссылка\]](#), а также в Википедии (в статьях про Округление или Rounding).
- Если все коэффициенты многочлена равны 0 или 0.0, то он выводится в виде строки из одного символа 0.

Таблица символов (указаны значения, от которых надо взять `chr` для получения соответствующей *цифры* в верхнем регистре):

x^0 8304
 x^1 185
 x^2 178
 x^3 179
 x^4 8308
 x^5 8309
 x^6 8310
 x^7 8311
 x^8 8312
 x^9 8313

Input	Output
<code>print(Poly(range(11)))</code>	$10x^{10} + 9x^9 + 8x^8 + 7x^7 + 6x^6 + 5x^5 + 4x^4 + 3x^3 + 2x^2 + x$
<code>print(Poly((1.2345, 0.0001, -1)))</code>	$-x^2 + 0.0x + 1.234$
<code>print(Poly((1, 0, 1, 0, -1, 0, 1)))</code>	$x^6 - x^4 + x^2 + 1$

Указание: Нужно реализовать функцию, которая по степени и коэффициенту выведет строковое представление монома данной степени. Для хранения юникод-символов можно использовать словарь `{'0': chr(8304), ...}`. Далее создать список строковых представлений мономов. У старшего коэффициента поправить знак (убрать $+$ и пробел перед минусом). После чего склеить вместе.

Е. Равенство, неравенство, `__neg__`, `__pos__`, `__bool__`

Реализуйте методы `__eq__`, `__ne__`, `__neg__`, `__pos__` и `__bool__`. Метод `__neg__` должен возвращать новый многочлен, равный минус текущему. Метод `__pos__` должен возвращать `self`. Метод `__bool__` возвращает `False` тогда и только тогда, когда многочлен равен нулю и возвращать `True` в противном случае.

Input	Output
<pre>A = Poly(range(5)) B = Poly((0, 1, 2, 3, 4)) C = Poly(range(4)) print(+A) print(-A) print(+A is A) print(-A is A) print(A == B) print(A == C) print(A != B) print(A != C) print(Poly(3) == 3) print(bool(Poly())) print(bool(Poly(1)))</pre>	<pre>4x⁴ + 3x³ + 2x² + x -4x⁴ - 3x³ - 2x² - x True False True True False False False True True True False True</pre>

Ф. Сложение многочленов

Реализуйте методы для сложения многочленов друг с другом, а также с числами типа `int` и `float`. Кроме `__add__` и `__radd__` необходимо также реализовать метод `__iadd__`, который используется в выражениях `(A+=2)`. Метод `__iadd__` должен модифицировать собственный набор коэффициентов и возвращать `self`. В классе рациональных чисел мы не стали реализовывать этот метод, так как они неизменяемы, при операции `A+=2` в `A` попадает уже новое рациональное число.

Input	Output
<pre>A = Poly(range(5)) B = Poly((1.23, 2.34, 0, -5)) C = Poly((0, 0, 0, 0, 0, 0, 2)) print(A + B) print(A + C) print(B + C) print(A + (-A)) print(A + B + C) print(3 + A) print(A + 3) print(2.1 + B) print(B + 2.1) A += 1 B += 1.23 D = C C += B + A print(C) print(D) print(C is D)</pre>	<pre>4x⁴ - 2x³ + 2x² + 3.34x + 1.23 2x⁶ + 4x⁴ + 3x³ + 2x² + x 2x⁶ - 5x³ + 2.34x + 1.23 0 2x⁶ + 4x⁴ - 2x³ + 2x² + 3.34x + 1.23 4x⁴ + 3x³ + 2x² + x + 3 4x⁴ + 3x³ + 2x² + x + 3 -5x³ + 2.34x + 3.33 -5x³ + 2.34x + 3.33 2x⁶ + 4x⁴ - 2x³ + 2x² + 3.34x + 3.46 2x⁶ + 4x⁴ - 2x³ + 2x² + 3.34x + 3.46 True</pre>

Г. Вычитание многочленов

Реализуйте методы для вычитания многочленов и чисел.

Input	Output
<pre>A = Poly(range(5)) B = Poly((1.23, 2.34, 0, -5)) C = Poly((0, 0, 0, 0, 0, 0, 2)) print(A - B) print(A - C) print(B - C) print(A - (+A)) print(A - B + C) print(3 - A) print(A - 3) print(2.1 - B) print(B - 2.1) A -= 1 B -= 1.23 D = C C -= B - A print(C) print(D) print(C is D)</pre>	<pre>4x⁴ + 8x³ + 2x² - 1.34x - 1.23 -2x⁶ + 4x⁴ + 3x³ + 2x² + x -2x⁶ - 5x³ + 2.34x + 1.23 0 2x⁶ + 4x⁴ + 8x³ + 2x² - 1.34x - 1.23 -4x⁴ - 3x³ - 2x² - x + 3 4x⁴ + 3x³ + 2x² + x - 3 5x³ - 2.34x + 0.87 -5x³ + 2.34x - 0.87 2x⁶ + 4x⁴ + 8x³ + 2x² - 1.34x - 1.0 2x⁶ + 4x⁴ + 8x³ + 2x² - 1.34x - 1.0 True</pre>

Н. Умножение многочленов

Реализуйте методы для умножения многочленов и чисел. Результат умножения многочлена на число, равное нулю — многочлен, тождественно равный *целому* нулю.

Input	Output
<pre>A = Poly(range(5)) B = Poly((1.23, 2.34, 0, -5)) C = Poly((0, 0, 0, 0, 0, 0, 2)) print(A * B) print(A * C) print(B * C) print(A * (+A)) print(A * B * C) print(3 * A) print(A * 3) print(2.1 * B) print(B * 2.1) A *= 1 B *= 1.23 D = C C *= B * A print(C) print(D) print(C is D)</pre>	<pre>-20x⁷ - 15x⁶ - 0.64x⁵ + 6.94x⁴ + 8.37x³ + 4.8x² + 1.23x 8x¹⁰ + 6x⁹ + 4x⁸ + 2x⁷ -10x⁹ + 4.68x⁷ + 2.46x⁶ 16x⁸ + 24x⁷ + 25x⁶ + 20x⁵ + 10x⁴ + 4x³ + x² -40x¹³ - 30x¹² - 1.28x¹¹ + 13.88x¹⁰ + 16.74x⁹ + 9.6x⁸ + 2.46x⁷ 12x⁴ + 9x³ + 6x² + 3x 12x⁴ + 9x³ + 6x² + 3x -10.5x³ + 4.914x + 2.583 -10.5x³ + 4.914x + 2.583 -49.2x¹³ - 36.9x¹² - 1.574x¹¹ + 17.072x¹⁰ + 20.59x⁹ + 11.808x⁸ + 3.026x⁷ -49.2x¹³ - 36.9x¹² - 1.574x¹¹ + 17.072x¹⁰ + 20.59x⁹ + 11.808x⁸ + 3.026x⁷ True</pre>

I. Возведение в степень

Реализуйте методы для быстрого возведения в степень. Показатель степени — целое неотрицательное число. Должны быть определены операторы `**` для левого операнда типа `Poly`, правого операнда типа `int` (при этом неотрицательное). Должен быть определен оператор `**=` для левого операнда типа `Poly`, правого операнда типа `int`.

Input	Output
<pre>A = Poly((1, 1)) B = Poly((2)) C = Poly((1, 0, 0, 0, 0, 0, 0, 0, 2.34, 0, 0, 0, -2)) print(A ** 0) print(A ** 1) print(A ** 2) print(A ** 10) print(B ** 100) print(C ** 3) D = C C **= 3 print(C) print(D) print(C is D) print(7 * X**10 - 3 * X**5 + (X - 3) * (2 - 5 * X**2))</pre>	<pre>1 x + 1 x² + 2x + 1 x¹⁰ + 10x⁹ + 45x⁸ + 120x⁷ + 210x⁶ + 252x⁵ + 210x⁴ + 120x³ + 45x² + 10x + 1 1267650600228229401496703205376 -8x³⁶ + 28.08x³² - 32.854x²⁸ + 24.813x²⁴ - 28.08x²⁰ + 16.427x¹⁶ - 6.0x¹² + 7.02x⁸ + 1 -8x³⁶ + 28.08x³² - 32.854x²⁸ + 24.813x²⁴ - 28.08x²⁰ + 16.427x¹⁶ - 6.0x¹² + 7.02x⁸ + 1 -8x³⁶ + 28.08x³² - 32.854x²⁸ + 24.813x²⁴ - 28.08x²⁰ + 16.427x¹⁶ - 6.0x¹² + 7.02x⁸ + 1 True 7x¹⁰ - 3x⁵ - 5x³ + 15x² + 2x - 6</pre>

J. Вычисление значения многочлена в точке

Реализуйте функцию вычисления значения выражения в точке x . Если `P` — объект класса `Poly`, `x` — объект типа `int` или `float`, то вычисление многочлена в точке `x` производится при помощи перегруженной операции $P|x$. Для перегрузки операции $P|x$ нужно определить метод `__or__` класса `Poly`. Левый операнд имеет тип `Poly`, правый операнд: тип `int` или `float`.

Значение многочлена в точке должно вычисляться по схеме Горнера.

Input	Output
<pre>A = Poly(range(5)) B = Poly((1, 1)) ** 10 print(A 1) print(A -1.0) print(B 0) print(B 1) print(B -1)</pre>	<pre>10 2.0 1 1024 0</pre>

К. Итератор по коэффициентам

Реализуйте метод-генератор `__iter__`, по очереди выдающий все коэффициенты, начиная с нулевого.

Input	Output
<pre>A = Poly(range(10)) print([x ** 2 for x in A]) print(*A)</pre>	<pre>[0, 1, 4, 9, 16, 25, 36, 49, 64, 81] 0 1 2 3 4 5 6 7 8 9</pre>

Л. Добавление поддержки класса *Fraction*

Добавьте во все методы поддержку встроенного в Python класса `Fraction`.

В методе `__str__` выводите дроби со знаменателем, не равным 1 в скобках (3/4). Дроби, со знаменателем, равным 1 выводите как целое.

Для использования класса `Fraction` в первой строчке вашей программы напишите

`from fractions import Fraction.`

Input
<pre>A = Poly((1, 1.2, Fraction(5, -2), 0, Fraction('-1.2'))) B = A ** 2 - 2 * A print(A) print(B)</pre>

Output
<pre>-(6/5)x⁴ - (5/2)x² + 1.2x + 1 (36/25)x⁸ + 6x⁶ - 2.88x⁵ + 6.25x⁴ - 6.0x³ + 1.44x² - 1</pre>

М. Деление многочленов с остатком: *divmod*

Реализуйте деление многочленов с остатком.

Пусть даны два многочлена A и B . При делении многочлена A на многочлен B получается два многочлена Q (частное) и R (остаток) такие, что $A = Q * B + R$, при этом $deg(R) < deg(B)$. Для нахождения частного и остатка используется алгоритм деления “в столбик”.

Необходимо реализовать метод `__divmod__(self, other)` — возвращает кортеж из двух элементов: частного и остатка. Для вызова используется `divmod(a, b)`.

Если в процессе деления возникает деление целого числа на целое, то результат деления должен иметь тип `Fraction`.

Input
<pre>A = Poly((1, 1)) ** 5 B = Poly((1, 1)) C = Poly((1, -1, 1)) print('{} = ({})*({}) + ({}).format(A, B, *divmod(A, B)) print('{} = ({})*({}) + ({}).format(A, C, *divmod(A, C)) print('{} = ({})*({}) + ({}).format(B, A, *divmod(B, A)) print('{} = ({})*({}) + ({}).format(B, 2, *divmod(B, 2))</pre>

Output
<pre>(x⁵ + 5x⁴ + 10x³ + 10x² + 5x + 1) = (x + 1) * (x⁴ + 4x³ + 6x² + 4x + 1) + (0) (x⁵ + 5x⁴ + 10x³ + 10x² + 5x + 1) = (x² - x + 1) * (x³ + 6x² + 15x + 19) + (9x - 18) (x + 1) = (x⁵ + 5x⁴ + 10x³ + 10x² + 5x + 1) * (0) + (x + 1) (x + 1) = (2) * ((1/2)x + (1/2)) + (0)</pre>

N. Деление многочленов с остатком: всё остальное

Необходимо реализовать:

- метод `__divmod__(self, other)` — возвращает кортеж из двух элементов: частного и остатка. Для вызова используется `divmod(a, b)`.
- метод `__rdivmod__(self, other)` — вызывается при делении числа на многочлен.
- операторы `//` и `%` для операндов типа `Poly`, `int`, `float`, `Fraction` при этом хотя бы один операнд имеет тип `Poly`.
- операторы `//=` и `%=` для левого операнда типа `Poly`, правого операнда типа `Poly`, `int`, `float`, `Fraction`.

Input	Output
<pre>A = Poly((1, 1)) ** 5 B = Poly((1, 1)) C = Poly((1, -1, 1)) print(A // B) print(B // A) print(A % C) print(C % A) A //= B print(A) A %= C print(A) print('{} = ({})*({}) + ({}).format(2, A, *divmod(2, A))</pre>	<pre>x⁴ + 4x³ + 6x² + 4x + 1 0 9x - 18 x² - x + 1 x⁴ + 4x³ + 6x² + 4x + 1 9x - 9 (2) = (9x - 9) * (0) + (2)</pre>

O. Определение длины многочлена и доступ к коэффициентам

Необходимо реализовать функцию определение длины многочлена `len(P)` и возможности чтения и изменения коэффициентов по индексу `P[i]`.

Для определения длины объекта необходимо определить метод `__len__(self)`. Данный метод должен возвращать длину списка коэффициентов, т.е. степень многочлена + 1.

Для возможности чтения значения коэффициента многочлена `P[i]` необходимо определить метод `P.__getitem__(self, i)`. Если значение `i` не является целым, необходимо генерировать исключение `IndexError` при помощи инструкции `raise IndexError()`. Исключение `IndexError` необходимо генерировать и в случае, когда $i < 0$.

Для возможности присваивания коэффициенту многочлена нового значения `val` необходимо определить метод `P.__setitem__(self, i, val)`. При этом если $i \geq \text{len}(P)$ необходимо “расширить” список коэффициентов до нужного значения, при $i < 0$ необходимо генерировать исключение `IndexError`.

Input	Output
<pre>A = Poly((1, 1)) ** 20 print(len(A)) print(A[0], A[1], A[2], A[10]) print(A[100])</pre>	<pre>21 1 20 190 184756 0</pre>

P. Композиция многочленов

Необходимо реализовать операцию композиции многочленов: если P и Q — объекты класса `Poly`, то $P(Q)$ должно возвращать объект класса `Poly`, являющийся композицией многочленов.

Для этого необходимо перегрузить определить метод `__call__(self, x)`. Если P — объект класса `Poly`, то при использовании объекта P , как вызова функции $P(x)$ будет вызван метод `P.__call__(x)`.

Метод `P.__call__(x)` должен возвращать значение многочлена в точке x , если x является объектом классов `int`, `float`, `Fraction`; если же x является объектом класса `Poly`, то метод `__call__` должен возвращать композицию многочленов.

Input	Output
<pre>A = Poly((1, 1)) B = A(A) print(B) print((A**2)(A**2)) print((A**10)(1))</pre>	<pre>x + 2 x⁴ + 4x³ + 8x² + 8x + 4 1024</pre>

Q. *Рациональные корни многочлена*

Реализуйте метод `rational_roots`, возвращающий отсортированный по неубыванию список рациональных корней многочлена с учётом кратности.

Если какой-либо коэффициент имеет тип, отличный от `int` или `Fraction`, то необходимо взвести ошибку `TypeError()`.

Input
<pre>print((X - 1) * (X - Fraction(3, 7)) * (X ** 2 + 2 * X + 2)).rational_roots() print(Poly((Fraction(9, 49), Fraction(-60, 49), Fraction(142, 49), Fraction(-20, 7), 1)).rational_roots())</pre>

Output
<pre>[Fraction(3, 7), 1] [Fraction(3, 7), Fraction(3, 7), 1, 1]</pre>

R. *Алгоритм Евклида*

Реализуйте функцию `pgcd(a, b)`, возвращающую НОД двух многочленов. Старший коэффициент НОДа должен быть равен 1. Если какой-либо коэффициент имеет тип, отличный от `int`, `Fraction`, то необходимо взвести ошибку `TypeError()`.

Input	Output
<pre>print(pgcd((X + 1)**10, (X + 1) * (X + 2)**2))</pre>	<pre>X + 1</pre>

Список методов, использующихся для перегрузки стандартных операций:

Операторы сравнения

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

Сложение

Метод	Использование
<code>__add__(self, other)</code>	<code>self + other</code>
<code>__radd__(self, other)</code>	<code>other + self</code>
<code>__iadd__(self, other)</code>	<code>self += other</code>

Вычитание

Метод	Использование
<code>__sub__(self, other)</code>	<code>self - other</code>
<code>__rsub__(self, other)</code>	<code>other - self</code>
<code>__isub__(self, other)</code>	<code>self -= other</code>

Умножение

Метод	Использование
<code>__mul__(self, other)</code>	<code>self * other</code>
<code>__rmul__(self, other)</code>	<code>other * self</code>
<code>__imul__(self, other)</code>	<code>self *= other</code>

Деление

Метод	Использование
<code>__truediv__(self, other)</code>	<code>self / other</code>
<code>__rtruediv__(self, other)</code>	<code>other / self</code>
<code>__itruediv__(self, other)</code>	<code>self /= other</code>

Целочисленное деление

Метод	Использование
<code>__floordiv__(self, other)</code>	<code>self // other</code>
<code>__rfloordiv__(self, other)</code>	<code>other // self</code>
<code>__ifloordiv__(self, other)</code>	<code>self //= other</code>
<code>__divmod__(self, other)</code>	<code>divmod(self, other)</code>

Остаток

Метод	Использование
<code>__mod__(self, other)</code>	<code>self % other</code>
<code>__rmod__(self, other)</code>	<code>other % self</code>
<code>__imod__(self, other)</code>	<code>self %= other</code>

Возведение в степень

Метод	Использование
<code>__pow__(self, other)</code>	<code>self ** other</code>
<code>__rpow__(self, other)</code>	<code>other ** self</code>
<code>__ipow__(self, other)</code>	<code>self **= other</code>

Отрицание, модуль

Метод	Использование
<code>__pos__(self, other)</code>	<code>+self</code>
<code>__neg__(self, other)</code>	<code>-self</code>
<code>__abs__(self, other)</code>	<code>abs(self)</code>