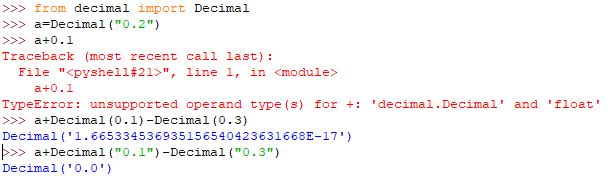
**Decimal**

Для того, чтоб использовать числа с фиксированной точностью, необходимо импортировать соответствующий встроенный модуль **decimal** с помощью оператора **import**, после которого указывается имя модуля. Позже рассмотрим это детальнее. Импортируем его командой:

from decimal import \*

Данный модуль содержит три концепции: десятичные числа (класс **Decimal)**, контекст для арифметики (это коллекция параметров настройки, определяющих поведение чисел decimal.Decimal) и сигналы исключений. Ключевым компонентом для работы с числами в этом модуле является класс **Decimal**. Для его использования необходимо создать объект с помощью конструктора. В конструктор передается строковое значение, которое представляет число:



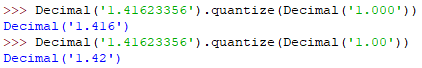
С помощью дополнительных знаков можно определить, сколько будет символов в дробной части числа:



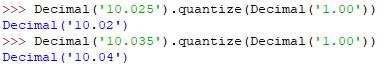
Объекты **Decimal** имеют много [встроенных методов](https://docs.python.org/3/library/decimal.html#decimal-objects). Рассмотрим некоторые их них.

Метод **quantize**

Метод **quantize**(exp, rounding=None, context=None) позволяет округлять числа. В этот метод в качестве первого аргумента передается также объект **Decimal**, который указывает формат округления числа:

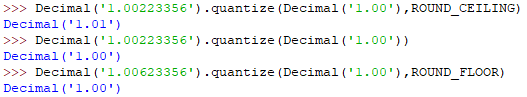


Второй параметр отвечает за поведение при округлении. По умолчанию округление числа происходит в большую сторону, если оно нечетное:



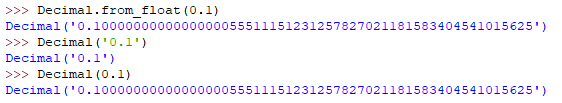
Чтоб изменить это, необходимо использовать один из следующих параметров:

* ROUND\_CEILING: округляет число в большую сторону вне зависимости от того, какое число идет после него
* ROUND\_FLOOR: не округляет число вне зависимости от того, какое число идет после него:



Доступны математические методы, такие как sqrt(), exp(), ln(), log10().

**from\_float(f) –** метод, который точно преобразует float в десятичное число. Стоит отметить, что ‘0.1’ и 0.1 это не одно и тоже, так как 0.1 не является точным представлением данного значения (представляется в двоичном виде с плавающей запятой).



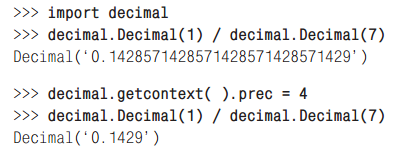
**max(other, context=None) –** максимальное из двух.

****

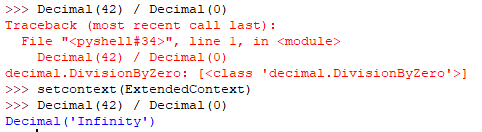
Как уже упоминалось, модуль **decimal** содержит контекстные объекты. Они регулируют точность, устанавливают правила округления, определяют, какие сигналы обрабатываются как исключения, и ограничивают диапазон для количества значащих цифр после запятой. Каждый поток управления имеет свой собственный текущий контекст, к которому можно обратится или изменить с помощью функций getcontext() и setcontext() :

**decimal.getcontext() –** возвращает текущий контекст для активного потока.

**decimal.setcontext(c) –** установка текущего контекста для активного потока в *c*.

****

Новые контексты могут быть созданы с использованием конструктора Context.



**class decimal.BasicContext**

Это стандартный контекст, определяемый общей десятичной арифметической спецификацией. Точность установлена ​​равной девяти. Для округления установлено значение ROUND\_HALF\_UP . Все флаги очищаются. Все «ловушки» для отлавливания ошибок включены, кроме Inexact , Rounded и Subnormal . Поскольку многие из ловушек включены, этот контекст полезен для отладки.

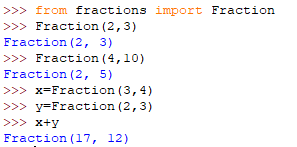
**class decimal.ExtendedContext**

Это стандартный контекст, определяемый общей десятичной арифметической спецификацией. Точность установлена ​​равной девяти. Для округления установлено значение ROUND\_HALF\_EVEN. Все флаги очищаются. Не включено отлавливание ошибок (так что исключения не возникают во время вычислений). Поскольку ловушки отключены, этот контекст полезен для приложений, которые предпочитают отображать значение результата NaN или Infinity вместо того, чтобы создавать исключения.

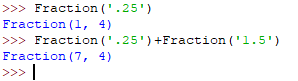
**Fraction**

Числовой тип – **Fraction** реализует объекты рациональных чисел. Объекты этого типа в явном виде хранят числитель и знаменатель рациональной дроби, что позволяет избежать неточности и некоторых других ограничений, присущих вещественным числам. Также, как и с decimal, для работы с **Fraction** необходимо импортировать модуль. Тип Fraction является своего рода родственником типа Decimal, описанного ранее, и точно так же может использоваться для управления точностью представления чисел за счет определения количества десятичных разрядов и политики округления.

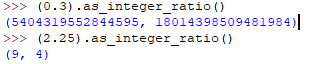
После создания объекты типа Fraction могут использоваться в математических выражениях как обычные числа:



Рациональные числа могут создаваться из строк с представлением вещественных чисел, как и числа с фиксированной точностью:



Для поддержки преобразования в рациональные числа в объектах вещественных чисел был реализован метод as\_integer\_ratio, возвращающий соответствующие числу числитель и знаменатель:



Однако стоить помнить, что несмотря на то, что имеется возможность преобразовать вещественное число в рациональное, в некоторых случаях это может приводить к потере точности, потому что в своем первоначальном виде вещественное число может быть неточным.

Таким образом, преимущество использования рациональных чисел в том, что они сохраняют точность и автоматически упрощают результат.

Cреди свободно распространяемых расширений можно найти и другие числовые типы. Например, матрицы и векторы.