Количество типов данных, которые имеются в любой системе компьютерной алгебры, значительно превышает количество типов данных в общеизвестных языках низкого уровня. По этой причине преобразование данных из одного типа в другой или же между различными представлениями носит гораздо более острый характер.
Пользователю было бы весьма трудно запомнить и активно применять на практике большое количество функций преобразования типов. К счастью, в системе Maple основную работу выполняет одна единственная функция *convert*. Рассмотрим основные преобразования, которые можно выполнить с помощью этой команды Maple.

В математике конечная десятичная дробь *всегда* считается точной величиной. Однако в системе Maple, как и во многих других языках программирования, наличие десятичной точки в записи числа указывает на то, что вычисления будут носить приближённый характер. Если же мы хотим придерживаться традиционной точки зрения и оперировать с десятичными дробями как с точными величинами, то эти дроби следует преобразовать в обыкновенные. Системы компьютерной алгебры трактуют рациональные числа как точные величины.

```
> convert(1.23456,rational);
```

$$\frac{3858}{3125}$$

Если выражение содержит большое количество операндов, которые представляют собой десятичные дроби, то многократное использование *convert* делает это выражение весьма громоздким и необозримым. В таком случае имеет смысл ввести оператор-функцию

```
> fr:=x->convert(x,rational);
```

$$fr := x \mapsto convert(x, rational)$$

Тогда предыдущий пример можно записать чуть короче

```
> fr(1.23456);
```

$$\frac{3858}{3125}$$

В контексте данного преобразования вместо типа данных *rational* можно написать *fraction*. Эти термины будут рассматриваться как синонимы.

```
> convert(1.23456,fraction);
```

$$\frac{3858}{3125}$$

Поскольку вычисления с вещественными числами могут выполняться с очень большой точностью, то алгоритм преобразования десятичной дроби в обыкновенную не столь тривиален, как на первый взгляд может показаться.
Рассмотрим сначала более простую ситуацию

```
> convert(0.33333,rational);
  convert(0.333333,rational);
```

$$\frac{33333}{100000}$$

$$\frac{333333}{1000000}$$

Создаётся впечатление, что преобразование сводится к умножению десятичной дроби на величину 10^{δ}, где δ – количество разрядов в дробной части числа. Так находится числитель обыкновенной дроби. В свою очередь, знаменатель будет всегда равен 10^{δ}. Однако попробуем увеличить количество разрядов

```
> convert(0.333333333,rational);
  convert(0.3333333333,rational);
```

$$\frac{333333333}{1000000000}$$

$$\frac{1}{3}$$

Как нам следует понимать полученный результат?

Оказывается, что при запуске сеанса работы в системе Maple инициализируется целый ряд системных переменных. Одна из таких переменных отвечает за точность выполнения операций с плавающей точкой. Она называется *Digits* и по умолчанию принимает значение 10.
Таким образом, если количество разрядов в дробной части числа, которое является аргументом функции *convert*, меньше чем *Digits*, то используется интуитивно напрашивающееся преобразование. Если же количество разрядов станет равным или же превысит *Digits*, то считается, что аргументом *convert* является бесконечная периодическая десятичная дробь и применяется совсем другой механизм преобразования

```
> convert(0.1515151515,rational);
```

$$\frac{5}{33}$$

Необдуманные манипуляции с глобальными переменными всегда опасны. Изменение *Digits* способно привести к неконтролируемой потере точности, поскольку некоторые команды Maple используют её для выполнения своих внутренних расчётов. Локальное изменение точности достигается за счёт использования третьего параметра функции *convert*, который тоже называется digits.
Если мы знаем, что наша десятичная дробь является точным значением, то мы пишем

```
> convert(1.6666,rational);
```

$$\frac{8333}{5000}$$

Когда мы хотим сказать, что дробная часть является бесконечной периодической дробью, т.е. наше число представимо в виде $1.666666666666666666666666\cdots$, то

```
> convert(1.6666,rational,4);
```

$$\frac{5}{3}$$

Если независимо от величины *Digits* мы хотим полностью заблокировать алгоритм перевода бесконечных десятичных дробей в обыкновенные, то третьим параметром *convert* следует написать слово exact. В таком случае всегда будет использоваться напрашивающийся способ преобразования.
(Сравните с приведённым выше примером.)

```
> convert(0.1515151515,rational,exact);
```

$$\frac{303030303}{2000000000}$$

Древние предания гласят, что великий Архимед, живший в III веке до нашей эры, с колоссальной для того времени точностью сумел оценить значение числа π, представив его в виде рациональной дроби равной $\frac{22}{7}$. Применение функции *convert* позволяет повторить результат Архимеда

```
> convert(evalf(Pi),rational,3);
```

$$\frac{22}{7}$$

В некоторых случаях приходится сталкиваться и с обратной задачей, когда рациональное число нужно представить в виде конечной десятичной дроби. По сути дела эта ситуация имитирует расчёты в режиме калькулятора. Она может пригодиться при проверке правильности промежуточных результатов. Функция *convert* успешно справляется и с этой задачей, если в качестве второго параметра написать слово float.

```
> convert(5/6,float);
```

$$0.8333333333$$

Количество разрядов в дробной части числа определяется текущим значением переменной *Digits*. Если нас это не устраивает, то в качестве третьего параметра *convert* следует указать желаемую точность выполнения операций с плавающей

точкой. Например, в школе часто число π заучивают с небольшой точностью.

```
> convert(Pi,float,3);
```

$$3.14$$

Локальные вычисления можно проводить с точностью превосходящей *Digits*.

```
> convert(exp(1),float,16);
```

$$2.718281828459045$$

Подобная возможность обусловлена тем, что *convert* с опцией float является "обёрткой" для команды *evalf*.

В системе Maple имеются средства, которые в ряде простейших случаев позволяют на основе десятичной дроби определить и саму величину, порождающую такое десятичное представление. Возьмём, например, сумму двух радикалов

```
> decFrac:=convert(2*sqrt(2)+sqrt(3),float);
```

$$decFrac := 4.560477932$$

Функция *identify* способна восстановить из десятичной дроби исходное математическое выражение.

```
> identify(decFrac);
```

$$2\sqrt{2}+\sqrt{3}$$

По этому поводу не следует питать слишком больших иллюзий, поскольку в менее тривиальных случаях требуется виртуозное знание опций команды *identify*.

По сложившейся традиции аргументы тригонометрических функций принято выражать в радианах. Однако на практике гораздо чаще приходится иметь дело с углами, измеренными в градусах. Для преобразования углов также следует использовать команду *convert*.

```
> convert(15*degrees,radians);
```

$$\frac{\pi}{12}$$

Обратите внимание, что, как старая, так и новая, единицы измерения записываются во множественном числе!

Ещё одна часто встречающаяся ошибка состоит в том, что пользователь забывает указать *знак умножения* между численным значением угла и его единицей измерения. В таком случае Maple выводит сиреневым цветом сообщение об ошибке.

```
> convert(15degrees,radians);
Error, missing operator or `;`
```

Правда для начинающего пользователя это сообщение может показаться не слишком информативным.

Если необходимо вычислить достаточно длинное выражение с большим количеством тригонометрических функций, аргумент которых выражен в градусах, то можно воспользоваться упомянутым ранее советом и ввести новую функцию-оператор.

```
> dr:=phi->convert(phi*degrees,radians);
```

$$dr := \phi \mapsto convert(\phi\ degrees, radians)$$

Использование такой функции не слишком сильно удлинит исходное выражение. Например, преобразование угла в 30° будет выглядеть так

```
> dr(30);
```

$$\frac{\pi}{6}$$

Очевидно, что существует и обратное преобразование, позволяющее пересчитать радианы в градусы. В этом преобразовании уже подразумевается, что исходный угол задан в радианах. Единица измерения результирующего угла по-прежнему должна указываться, причём *во множественном числе*.

```
> convert(Pi,degrees);
```

$$180\ degrees$$

Если известно численное значение угла, то преобразованный результат

с практической точки зрения будет выглядеть не слишком полезным

```
> convert(1,degrees);
```

$$\frac{180\ degrees}{\pi}$$

Возвращаясь к написанному ранее, мы убеждаемся в том, что можем достичь желаемой цели с помощью вложенного вызова *convert*.

```
> convert(convert(1,degrees),float);
```

$$57.29577950\ degrees$$

Подобная точность может показаться вам избыточной. Тогда предыдущую команду следует подкорректировать в соответствии с указанными выше рекомендациями.

```
> convert(convert(1,degrees),float,4);
```

$$57.29\ degrees$$

Обратите внимание, что при таких расчётах не гарантируется корректное округление последнего значащего разряда результата.

О системах счисления

В школьном курсе информатики отмечалась важность умения преобразовывать числа из одной позиционной системы счисления в другую. Для понимания особенностей выполнения арифметических операций на современных компьютерах особый интерес представляет двоичная система счисления. Функция *convert* успешно справляется с поставленной задачей, если в качестве второго аргумента указать ей параметр binary.

```
> convert(10,binary);
```

$$1010$$

Результатом этого вызова является двоичное число, записанное в привычной позиционной системе счисления (младшие разряды находятся справа). Первый аргумент данной команды *всегда* интерпретируется как десятичное число. При необходимости оно вычисляется.

```
> convert(2*3,binary);
```

$$110$$

Отрицательные числа преобразуются как положительные, а перед результатом ставится знак.

```
> convert(-10,binary);
```

$$-1010$$

Об этой особенности следует помнить, поскольку в памяти компьютера отрицательные числа хранятся в виде *дополнения до 2*.

Полезным свойством *convert* является способность этой функции преобразовывать к двоичной форме записи не только целые числа, но и десятичные дроби.

```
> convert(1.2,binary);
```

$$1.001100110$$

В двоичной системе счисления конечная десятичная дробь, как правило, становится бесконечной. Количество отображаемых двоичных разрядов определяется текущим значением переменной *Digits*. Если нас не устраивает стандартная точность, то, как и в случае с преобразованием float, можно изменить локальный контекст, указав в команде ещё один аргумент.

```
> convert(1.2,binary,16);
```

$$1.0011001100110011$$

По исторических причинам (системе Maple уже более 30 лет) команда *convert* имеет специальные опции для систем счисления, которые применялись ранее для описания архитектур некоторых компьютеров. Одна из таких систем счисления − восьмеричная.

```
> convert(26,octal);
```

$$32$$

При использовании опции octal, так же как и в случае binary, считается, что первый аргумент команды *convert* является десятичным числом. Результат отображается в виде числа, записанного в позиционной системе счисления слева направо, т.е.

старшие разряды расположены слева.

Сохраняются и другие аналогии с опцией binary. В частности, можно преобразовывать не только целые числа, но и десятичные дроби.

```
> convert(1.2,octal);
```

1.146314632

Точность преобразования также можно отрегулировать

```
> convert(1.2,octal,20);
```

1.1463146314631463146

В то же время не стоит полностью полагаться на встроенную документацию Maple. В описании этой функции утверждается, что её можно применять и для отрицательных чисел. Однако выполнение следующей команды приводит к сообщению об ошибке.

```
> convert(-64,octal);
```

```
Error, invalid input: `convert/octal` expects its 1st argument, n, to be of type And(nonnegative, {float, integer}), but received -64
```

В компьютерной технике достаточно популярной является шестнадцатеричная система счисления. Она позволяет сгруппировать 4 двоичных бита в одну "цифру". Для этой системы у функции *convert* также предусмотрена отдельная опция hex.

```
> convert(100,hex);
```

64

Вместо hex можно использовать синоним hexadecimal. Результат будет таким же.

```
> convert(100,hexadecimal);
```

64

"Цифры" шестнадцатеричной системы счисления, не совпадающие с десятичными, выводятся заглавными буквами латинского алфавита.

```
> convert(156,hexadecimal);
```

9C

В отличие от рассмотренных ранее случаев, первый аргумент должен представлять собой целое неотрицательное число. Иначе возникает ошибка.

```
> convert(15.6,hexadecimal);
```

```
Error, invalid input: `convert/hex` expects its 1st argument, n, to be of type nonnegint, but received 15.6
```

Рассмотренные выше преобразования позволяли переводить числа *из* десятичной системы счисления в какую-либо другую. Было бы весьма удивительно, если бы не существовало обратного преобразования. Такое преобразование действительно существует. Для этого второй аргумент команды *convert* должен принимать значение decimal. В данной ситуации обязательным является третий аргумент. Он указывает на то, в какой системе счисления представлено исходное число. Основание системы счисления следует указывать либо в числовой форме, либо с помощью одной из следующих констант: binary, octal, hexadecimal или hex.

В качестве примера переведём число сначала в шестнадцатеричную систему,

```
> hNum:=convert(158,hexadecimal);
```

hNum := *9E*

а затем обратно.

```
> convert(hNum,decimal,hex);
```

158

Вместо символьной константы можно использовать числовое значение.

```
> convert(hNum,decimal,16);
```

158

Контроль входного числа на соответствие указанной системе счисления **отсутствует**. В результате интерпретация некоторых вызовов функции требует изобретательности.

```
> convert(189,decimal,octal);
```

137