

Примитивные типы данных и некоторые их функции

[Строковые константы](#)

[Числовые выражения](#)

[Булевские значения](#)

[Литералы типа Дата](#)

[Преобразования примитивных типов данных](#)

[Значения типа NULL и Неопределено](#)

[Тип данных Тип](#)

Выделяют следующие примитивные типы данных:

- строковые константы;
- числовые выражения;
- булевские значения (*Истина* и *Ложь*);
- литералы типа *Дата*;
- тип данных с единственным значением *Неопределено*;
- тип данных *NULL*, который также состоит из единственного значения;
- тип данных типа *Тип*.

Строковые константы

Примитивный тип данных *Строка* (строковая константа) состоит из различных символов. Строка всегда обрамляется кавычками. Пример строковой константы:

```
Сообщение.Текст = "Присутствуют незаполненные данные";
```

Т.е. строка *"Присутствуют незаполненные данные"* присваивается реквизиту *Текст* объекта *Сообщение*. Все, что обрамлено в кавычки, считается строкой. Строка может состоять из любых символов. Строки могут быть многострочными. При этом каждую новую строку необходимо определять в кавычки. Например:

```
Текст = "Неверно заполнен реквизит"
```

```
"Проведение документа невозможно";
```

Точка с запятой ставится только в конце последней строки.

Существует еще один способ – весь текст обрамлять только в одни кавычки, но каждая новая строка должна начинаться с вертикальной полосы. Такой синтаксис наиболее часто используется в типовых конфигурациях. В частности, в языке запросов. Например:

```
Запрос.Текст =  
"ВЫБРАТЬ  
| Сотрудники.Наименование КАК Сотрудник,  
| Сотрудники.ДатаРождения КАК ДатаРождения  
| ИЗ  
| Справочник.Сотрудники КАК Сотрудники  
| ГДЕ  
| НЕ Сотрудники.ЭтоГруппа";
```

Следует отметить, что для строк определена операция сложения. Это не арифметическая операция, ее называют операцией конкатенации. Т.е. нужно объединить, например, две строки, при этом между строками ставится знак сложения «+»:

```
Текст = "Неверно заполнен реквизит" + "Проведение документа невозможно";
```

Таким образом, происходит склеивание строк. Операция конкатенации, естественно, применима и для большего количества строк. Другие операции (вычитание, умножение, деление) для строк не допустимы.

Если внутри строки какое-то слово нужно обрамлять в кавычки, то кавычку внутри строки нужно определять двойной кавычкой. Например:

```
Текст = "Ошибка в модуле ""Общий модуль1""";
```

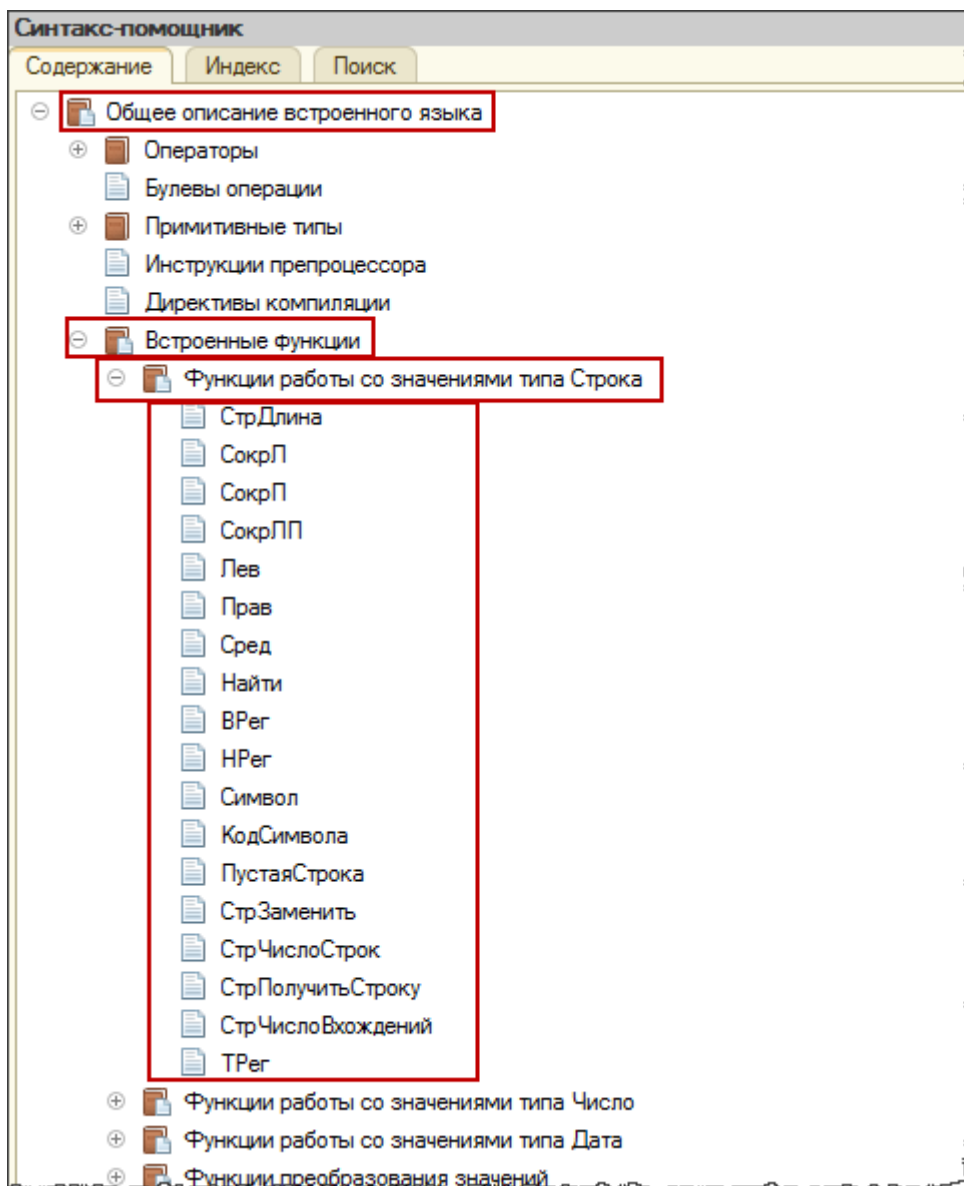
В данном примере первая кавычка открывает строку. Рядом стоящие вторая и третья кавычки обозначают знак кавычки. А в конце получается три кавычки: самая последняя кавычка закрывает строку, две предыдущие обозначают знак кавычки.

Над строками возможны различные операции преобразования строк, определение нескольких первых левых символов, определение нескольких крайних правых символов, поиск подстроки внутри строки и т.д.

Все эти функции доступны в любом месте конфигурации.

В синтакс-помощнике они находятся в разделе *Общее описание встроенного языка - Встроенные функции - Функции работы со значениями типа Строка.*

Функций достаточно большое количество и их обычно достаточно для работы со строковыми константами.



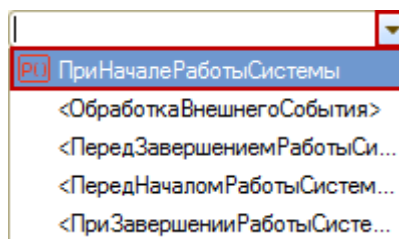
Разберем пример решения задачи с использованием строковых функций.

Условие задачи:

Требуется разработать функцию. В качестве параметра в функцию передается произвольная строка. Символами в строке могут быть в том числе и цифры.

Последовательность цифр (от одной и больше), ограниченная от других знаков пробелами, составляет целое положительное число. Например, строка "72 ABC 6АП 31 54ф -22" содержит два целых положительных числа: 72 и 31. Кроме пробелов, другие незначащие символы (такие как табуляция, возврат каретки) не используются. Функция должна возвращать количество целых положительных чисел.

Она должна размещаться в модуле управляемого приложения. Необходимо обеспечить ее вызов при запуске системы. Строку определить с помощью переменной. Итак, отроем модуль управляемого приложения и выберем в поле выбора из списка в панели конфигуратора *Модуль стандартный обработчик ПриНачалеРаботыСистемы()*.



Внутри обработчика определим переменную *Строка*, например:

```
Строка = "72 ABC 6АП 31 54ф -22";
```

Далее оформим вызов функции, которую предстоит разработать:

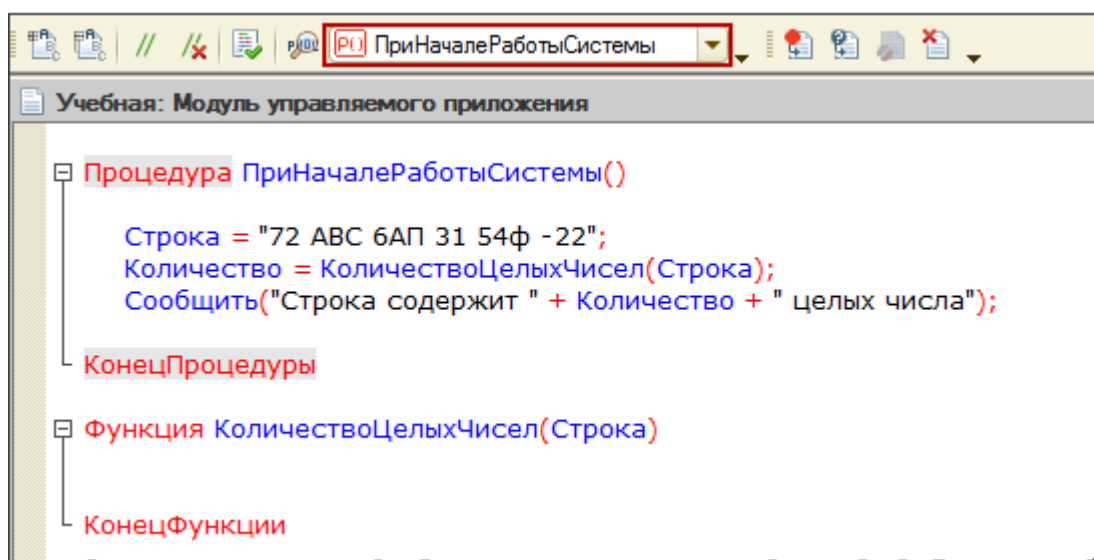
```
Количество = КоличествоЦелыхЧисел(Строка);
```

Оформим передачу сообщения о количестве целых чисел:

```
Сообщить("Строка содержит " + Количество + " целых числа");
```

При этом переменная *Количество* будет неявно преобразована к типу *Строковая константа*. Затем будет совершена операция конкатенации для трех строк и передано сообщение.

Определим начало и конец (т.е. шаблон) функции *КоличествоЦелыхЧисел(Строка)*



```
ПриНачалеРаботыСистемы
-----
Процедура ПриНачалеРаботыСистемы()
    Строка = "72 ABC 6АП 31 54ф -22";
    Количество = КоличествоЦелыхЧисел(Строка);
    Сообщить("Строка содержит " + Количество + " целых числа");
КонецПроцедуры

Функция КоличествоЦелыхЧисел(Строка)
КонецФункции
```

Теперь рассмотрим один из возможных вариантов разработки функции *КоличествоЦелыхЧисел(Строка)*. При этом познакомимся с некоторыми встроенными функциями, предназначенными для работы со строками.

Прежде всего, следует познакомиться с функцией *КодСимвола*. Эта функция получает код символа, расположенного в переданной строке в позиции с указанным номером.

Синтаксис:

КодСимвола (<Строка>, <НомерСимвола>)

Параметры:

<Строка> (обязательный)

<НомерСимвола> (необязательный) – это номер символа в строке, код которого необходимо получить. Нумерация символов в строке начинается с 1.

Возвращаемое значение:

Код переданного символа. Код возвращается в соответствии с кодировкой Unicode.

Обратите внимание, что у параметра <НомерСимвола> есть значение по умолчанию 1.

Строка тоже может состоять из одного символа. Таким образом, есть возможность определить код 0 и код 9, а коды всех остальных цифр находятся, как известно, в интервале между ними.

Определим соответствующие переменные и их значения:

```
Код0 = КодСимвола ("0");
```

```
Код9 = КодСимвола ("9");
```

Для решения задачи выберем следующую схему:

1. Если в строке присутствуют начальные или конечные пробелы в любом количестве, то избавимся от них специальной функцией. Далее нас будут интересовать группы символов между внутренними пробелами. Если группа состоит из одних цифр, то это целое число. Есть специальная функция, с помощью которой можно определить позицию первого пробела.
2. Получив позицию первого пробела, с помощью другой функции можно получить группу символов (подстроку) слева от пробела.
3. Проанализируем символы, составляющие группу и определим: является ли она целым числом. Выявленные целые числа будем суммировать в специальной переменной.
4. Укоротим начальную строку, выбрав с помощью еще одной функции все символы теперь уже справа от пробела. Данный пробел мог быть не один, а целая серия пробелов, идущих подряд, поэтому в оставшейся строке специальной функцией избавимся от всех крайних левых пробелов (идущих подряд) и вернемся к пункту 2. Будем повторять действия от пункта 2 до пункта 4, пока не достигнем состояния, что в строке не останется пробелов. В этом случае укороченная строка будет составлять последнюю группу анализируемых символов.

Теперь разберем функции, которые нам понадобятся для решения задачи.

СокрЛП

Синтаксис: СокрЛП (<Строка>)

Параметры: <Строка> (обязательный).

Отсекает пробелы (незначащие символы), стоящие слева от первого значащего символа в строке, и стоящие справа от последнего значащего символа в строке.

Найти

Синтаксис: Найти (<Строка>, <ПодстрокаПоиска>)

Параметры: <Строка> (обязательный), <ПодстрокаПоиска> (обязательный).

Возвращает позицию первого знака найденной подстроки. Нумерация символов в строке начинается с 1. Если строка не содержит указанной подстроки, то возвращается 0. В нашем случае в качестве подстроки будем использовать пробел (« »).

Лев

Синтаксис: Лев (<Строка>, <ЧислоСимволов>)

Параметры: <Строка> (обязательный), <ЧислоСимволов> (обязательный).

Выбирает первые слева символы строки. С помощью этой функции будем определять группы символов для анализа (слева до первого пробела).

СтрДлина

Синтаксис: СтрДлина (<Строка>)

Параметры: <Строка> (обязательный).

Получает количество символов в строке. Будем использовать для определения длины строки.

Функция *КодСимвола*, которую будем использовать для выявления групп символов, являющихся целыми числами, описана ранее.

Прав

Синтаксис: Прав (<Строка>, <ЧислоСимволов>)

Параметры: <Строка> (обязательный), <ЧислоСимволов> (обязательный).

Выбирает крайние справа символы строки. С помощью этой функции будем выделять еще необработанную часть строки.

СокрЛ

Синтаксис: СокрЛ (<Строка>)

Параметры: <Строка> (обязательный).

Отсекает пробелы (незначащие символы), стоящие слева от первого значащего символа в строке. Эту функцию используем для удаления возможных пробелов слева оставшейся части строки.

Ниже представлен возможный алгоритм функции с комментариями.

```
Конец = Ложь; // Конец будет равен Истине после того, как определится
//последняя группа анализируемых символов
Код0 = КодСимвола("0");
Код9 = КодСимвола("9");

Строка = СокрЛП(Строка); // Избавились от пробелов слева и справа
ДлинаСтроки = СтрДлина(Строка);
КоличествоЦелыхЧисел = 0;
ПозицияПробела = Найти(Строка, " "); // Определили позицию ближайшего пробела

Пока НЕ Конец Цикл

    Если ПозицияПробела > 0 Тогда
        ДлинаГруппыСимволов = ПозицияПробела-1;
        ГруппаСимволов = Лев(Строка, ДлинаГруппыСимволов); // Определили группу анализируемых символов

        Иначе // это последняя группа символов
            ГруппаСимволов = Строка;
            ДлинаГруппыСимволов = СтрДлина(ГруппаСимволов);
            Конец = Истина;
        КонецЕсли;

        // Ищем в группе хоть один символ, не являющийся цифрой
        Для индекс=1 По ДлинаГруппыСимволов Цикл
            Если КодСимвола(ГруппаСимволов,индекс) < Код0 или КодСимвола(ГруппаСимволов,индекс) > Код9 Тогда
                Прервать;
            КонецЕсли;
        КонецЦикла;

        // Если в группе все цифры, то индекс будет больше длины подстроки на 1
        Если индекс > ДлинаГруппыСимволов Тогда // В этом случае найдено целое число
            КоличествоЦелыхЧисел = КоличествоЦелыхЧисел + 1;
        КонецЕсли;

        Строка = Прав(Строка, ДлинаСтроки-ПозицияПробела); // Укоротили строку, оставив символы справа
        //от пробела
        Строка = СокрЛ(Строка); // Избавились от возможных пробелов слева
        ДлинаСтроки = СтрДлина(Строка);
        ПозицияПробела = Найти(Строка, " ");
    КонецЦикла;

Возврат КоличествоЦелыхЧисел;
```


Числовые выражения

Числовыми могут быть переменные модулей и реквизиты объектов базы данных. Для числа существует ограничение разрядности. Для числового реквизита длина целой части не может превышать 32 символов.

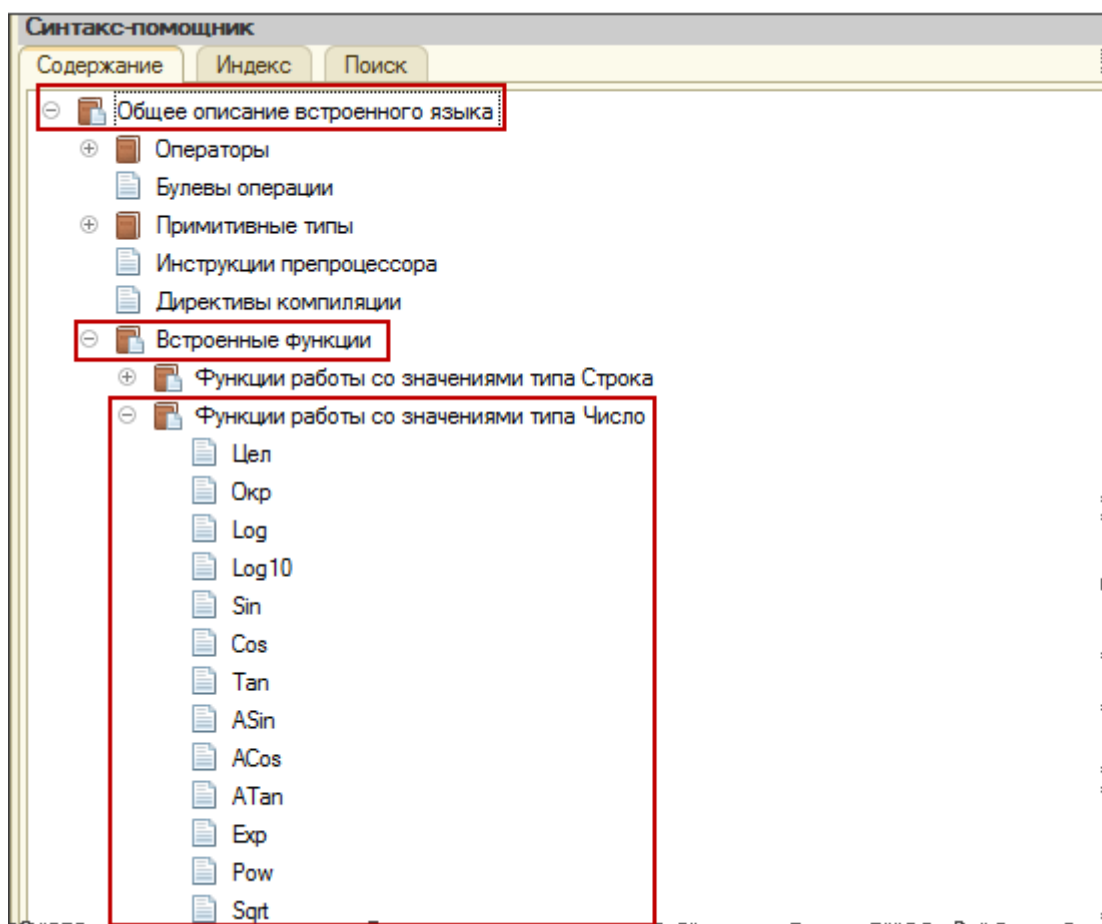
Точность дробной части не может превышать 10 цифр. Когда описывается переменная и присваивается ей числовое значение, то нигде не фиксируется ее разрядность. Тем не менее, для переменных тоже существуют ограничения. В синтаксис-помощнике сказано, что максимально допустимая разрядность для числа – это 38 знаков.

Такое ограничение не препятствует решению любых экономических задач, т.е. любую денежную величину можно описать с помощью этих чисел.

Тем не менее, если все-таки потребуются описать большие величины для решения каких-то математических задач, то в теории программирования есть алгоритмы, позволяющие описать числа с любой размерностью на основании существующих ограничений.

Операции, применимые для чисел:

- обычные арифметические операции ($-$, $+$, $*$, $/$). У умножения и деления приоритет больше, чем у сложения и вычитания. Скобки имеют наивысший приоритет. Есть еще унарные операции $+$ и $-$, у которых приоритет идет сразу за скобками;
- операция “остаток от деления” ($\%$). Например, $12\%5=2$;
- математические функции, которые можно применять для чисел (тригонометрические функции, возведение в степень, извлечение квадратного корня, округление до указанной разрядности, выбор целой части числа).



Если говорить о точности числовых значений, что касается реквизитов базы данных, то здесь присутствуют естественные ограничения.

Но что касается переменных, то здесь есть особенность. На самом деле, в переменных можно оперировать очень большими числами, но в информационную базу будут сохраняться значения с длиной целой части не больше 32 знаков.

Булевские значения

Что касается типа данных Булево, то здесь существует только два значения *Истина* и *Ложь*, которые могут быть получены различными способами.

Можно, например, использовать операции сравнения чисел или дат. В итоге будет получаться некое булевское значение, которое в дальнейшем наиболее часто используется в условных операторах и в операторах цикла.

Литералы типа Дата

Для описания даты существует два способа. Один из них с использованием литерала. Литерал пишется в одинарных кавычках. Сначала пишется год, потом месяц и затем день.

При необходимости, можно указать и время, т.к. в системе 1С:Предприятие 8 любая дата содержит в себе и дату и время. Например:

```
ДатаДокумента = ' 20140315121020';
```

Если время не указано, то по умолчанию оно равно нулю. В описании даты можно использовать любой разделитель. Например:

```
ДатаДокумента =' 2014.03.15';
```

Второй способ определения даты - это использование функции глобального контекста *Дата(<Значение>)*. В этом случае мы передаем в качестве параметров этой функции тоже самое: год, месяц, день через запятую.

Также можно указать время. Если его не указывать, тогда оно будет равно по умолчанию началу дня.

В системе 1С:Предприятие 8 пустая дата – это самое начало календаря. Варианты записи:

```
ПустаяДата = '00010101';
```

```
ПустаяДата = Дата(1,1,1);
```

И та и другая запись вернет одинаковый результат, и эта дата будет считаться пустой.

Удобство функции *Дата(<Значение>)* в том, что мы можем передавать в нее не конкретные значения, а какие-то переменные. Т.е., иногда мы дату конструируем, собирая разные переменные.

Для даты применима операция сложения. Операция сложения прибавляет к дате указанное количество секунд.

Преобразования примитивных типов данных

В операторе присваивания, где суммируется несколько переменных (например, Переменная = A + B + C), возможно преобразование примитивных типов данных. Преобразование типа данных осуществляется по значению первого типа данных.

Таким образом, если первый тип данных строка, то система будет пытаться сделать из всего этого выражения строку. Если первый тип данных число, то, соответственно, система попытается получить числовой тип данных.

И так, строка + число = строка. Иногда число можно сложить со строкой, в том случае, если из строки можно выделить какое-то числовое значение (например, 123 + "456").

Для логического типа данных применимы выражения:

Истина И 1 = Истина;

Истина И 0 = Ложь.

Любое число больше нуля преобразуется в *Истина*, 0 преобразуется в *Ложь*.

Дату можно, как отмечалось раньше, складывать с числом. Дату можно также складывать с булевым типом данных. В этом случае *Истина* преобразуется в 1, а *Ложь* в 0.

Кроме преобразования типов в операторах возможно явное преобразование типов с использованием соответствующих функций: *Строка(<Значение>)*, *Число(<Значение>)*, *Дата(<Значение>)*, *Булево(<Значение>)*.

К *Строке* конвертируется любой тип данных.

Число может быть получено из *Строки* или из *Булево*. *Булево* конвертируется: *Истина* в 1, *Ложь* в 0.

К *Дате* можно привести строку, если там будет содержаться значение даты. Например, *Дата("20140315")*. Как отмечалось ранее, возможно преобразование по позициям:

Дата (<Год>, <Месяц>, <День>, <Час>, <Минута>, <Секунда>).

В *Булево* можно преобразовать *Число* и само значение *Булево*.

Данные функции можно использовать в программном коде для того, чтобы выполнить явное преобразование типов.

Примитивные типы данных *Число*, *Строка*, *Дата* и *Булево* могут выступать в качестве полей базы данных.

Значения типа `NULL` и Неопределено

`NULL` – это литерал. Применяется он, как правило, в запросах к базе данных, когда соединяются две и более таблиц.

Именно отсутствующие записи во второй таблице и заполняются значением типа `NULL`. Т.е. это некое отсутствующее значение. В дальнейшем, при обработке получившегося результата это нужно учитывать, так как `NULL` – это не ноль, а соответствующий тип данных.

Для того, чтобы значение обработать, нужно `NULL` привести к какому-либо обычному типу данных, который можно выводить или использовать в арифметических операциях.

Значение типа `NULL` можно получить и во встроенном языке. Можно определить некоторую переменную и присвоить ей это самое значение `NULL`. Однако подобное присваивание в программном коде практически никогда не используется.

Т.е. `NULL` – это действительно тот тип данных, который получается при работе с запросами. Значение `NULL` именно на языке запросов нужно обрабатывать по-особенному. А именно, на уровне запросов не будет работать сравнение `A=NULL`, нужно будет применять специализированные функции. Однако во встроенном языке сравнение со значением `NULL` будет корректно отработано.

Тип данных *Неопределено* – это не пустое значение какого-либо реквизита. Например, если реквизит справочника имеет в качестве типа данных ссылку на какой-либо другой справочник, то пустое значение этого реквизита не будет равно *Неопределено*.

Данный тип (*Неопределено*) появляется, во-первых, если у нас есть некая переменная и она не инициализирована (тип данных не определен).

Второй пример: тип данных *Неопределено* возвращают многие функции встроенного языка, если действие не может быть выполнено. Например, поиск элемента справочника по наименованию в том случае, если у какого-либо справочника нет такого наименования элемента. Метод *НайтиПоНаименованию* будет возвращать значение *Неопределено*.

При этом *Неопределено* является ключевым словом, оно подсвечивается красным цветом. Это тоже литерал, для написания *Неопределено* не нужно использовать никаких кавычек, запятых, скобок и т.д.

Если имеется список документов, и этот список пустой (в нем, соответственно нет ни одной строки), то текущая строка будет принимать значение *Неопределено*.

Если в информационной базе есть реквизит с составным типом данных, то пустое значение данного реквизита будет равно *Неопределено*. Но если тип данных будет не составным, то пустое значение будет соответствовать пустому значению данного типа (для даты – это первая секунда первого часа первого дня первого месяца первого года).

NULL и *Неопределено* – это и типы данных и значения в этих типах, причем одно единственное. Для *NULL* – это значение *NULL*, для *Неопределено* – *Неопределено*.

Тип данных Тип

Основное применение этого типа данных заключается в том, чтобы сравнить значение некой переменной или реквизита базы данных с конкретным типом. Т.е. в алгоритме нужно понять, какой тип у данного объекта.

Примечательно, что этот тип данных не имеет литерала. Мы не можем его написать, как например, *NULL* или *Неопределено*, но мы можем получить значение этого типа с помощью двух функций *Тип* и *ТипЗнч*.

Для того, чтобы получить тип некоторого объекта (это может быть переменная, либо реквизит базы данных, либо реквизит формы), используется функция *ТипЗнч*.

В эту функцию передается тот объект, для которого требуется получить тип данных.

В качестве возвращаемого значения эта функция возвращает именно тип типа тип. В дальнейшем следует его сравнить с каким-либо интересующим типом. Например:

```
Если ТипЗнч(Элемент) = Тип ("СправочникСсылка.Номенклатура") Тогда
```

```
Сообщить ("Это товар");
```

```
КонецЕсли;
```

Артемов Артем,

г. Москва

Дополнительные материалы

Все статьи проекта Курсы-по-1С.рф: <http://курсы-по-1с.рф/blog/articles/>

Курсы по программированию в 1С v.8

Базовый и Продвинутой курсы по Программированию на Платформе 1С 8
<http://www.Spec8.ru/>



Базовый курс по программированию в 1С v.8

Курс про **готовые приемы и решения**
90% задач по программированию в 1С



Продвинутой курс по программированию в 1С v.8

Больше, чем Вы можете себе представить
Детальнее требований на **1С:Специалист**

«Курс по подготовке к Аттестации по Платформе 1С 8.2 / 8.3»

<http://курсы-по-1с.рф/dev-attestation/>



Подготовка к Аттестации по Платформе 1С v.8

Аттестация по Платформе – **с первого раза**
Экономия 100 - 150 часов подготовки