УВАЖАЕМЫЕ СТУДЕНТЫ! Законспектируйте в своей рабочей тетради по дисциплине приведенную лекцию (объемом 4-5 страницы), ответьте *письменно* на контрольные вопросы.

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: igor-gricenko-95@mail.ru в течении ТРЕХ дней.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)132-63-42

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 20

Тема: Ввод-вывод информации, переменные в JavaScript.

Цель: Изучить основные переменные в JavaScript.

Ввод и вывод данных

Сценарии на JavaScript могут взаимодействовать с объектами браузера и загруженного в него документа. Для ввода и вывода данных можно воспользоваться методами этих объектов. Каких-то специфических собственных методов вывода данных у JavaScript, интерпретируемого Web-браузерами, не предусмотрено.

Объект, представляющий свойства браузера, называется window (окно), а три его метода — alert(), prompt() и confirm() — предназначены для ввода и вывода данных посредством диалоговых окон (панелей). Поскольку это методы объекта window, то для их вызова, согласно каноническим правилам, следует писать window.alert(), window.prompt() и window.confirm() соответст- венно. Здесь используется так называемый точечный синтаксис, согласно которому перед обращением к свойству (переменной или методу) указывают соответствующий объект, за которым следует точка. Вместе с тем, поскольку объект window корневой в объектной модели, при обращении к его свойствам имя можно опустить. В этом случае интерпретатор и так поймет, что вызываемый метод относится к объекту window. Иначе говоря,

допускается сокращенная запись без префикса window. Например, если мы хотим вывести что-то, обозначенное через x, то можно написать в программе alert(x).

Внешний вид указанных диалоговых окон зависит от браузера. На рис. 1 показано, как они выглядят в Internet Explorer.

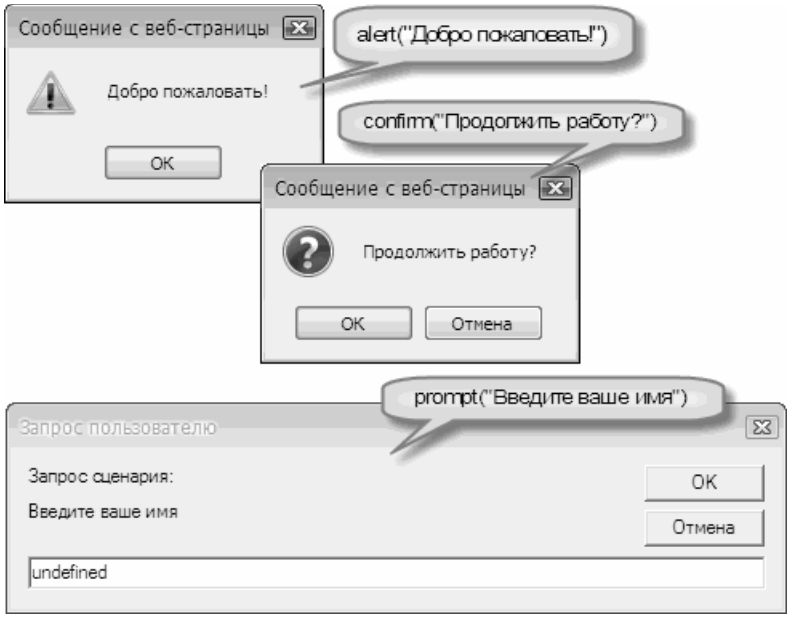

Рис. 1. Диалоговые окна, созданные методами alert(), confirm()и prompt()

Объект, представляющий свойства документа, называется document, а его методы write() и writeln() служат для вывода данных непосредственно в клиентскую область браузера, т. е. туда, где отображаются Web-страницы. Объект document очень важный, но не корневой в объектной модели, а потому при обращении к его свойствам требуется указывать имя объекта, например,

document.write("Привет!").

В следующих разделах мы рассмотрим перечисленные методы подробнее.

Метод *alert()*

Данный метод позволяет выводить диалоговое окно с заданным сообщением и кнопкой **OK**. В такое окно обычно выводят предупреждения. Его также удобно использовать для вывода промежуточных результатов при отладке скриптов. Синтаксис соответствующего выражения:

alert(*сообщение*).

Если ваше *сообщение* конкретно, т. е. представляет собой вполне определенный набор символов, то его необходимо заключить в двойные или одинарные кавычки: alert("Привет!"). Вообще говоря, *сообщение* представляет собой данные любого типа: последовательность символов, заключенную в кавычки, число (в кавычках или без них), переменную или выражение.

Диалоговое окно, выведенное на экран методом alert(), можно закрыть, щелкнув на кнопке **OK**. До тех пор, пока вы не сделаете этого, переход к ранее открытым окнам

невозможен. Окна, обладающие свойством останавливать все последующие действия пользователя и программ, называются *модальными*. Таким образом, метод alert()создает модальное окно.

Ранее уже отмечалось, что метод alert() пригоден для вывода промежуточных и окончательных результатов программ при их отладке. При этом можно отобразить результат вычисления какого-либо выражения и приостановить дальнейшее выполнение работы программы до тех пор, пока вы не нажмете кнопку **OK**.

Метод *confirm()*

Метод confirm позволяет вывести диалоговое окно с сообщением и двумя кнопками: **OK** и **Отмена** (Cancel). В отличие от alert(), этот метод возвращает логическую величину, значение которой зависит от того, какую из кнопок нажал пользователь. Если он щелкнул на кнопке **OK**, то возвращается значение true (истина, да); если же была нажата кнопка **Отмена** (Cancel), то возвращается значение false (ложь, нет). Возвращаемое значение можно обработать

в программе и, следовательно, создать эффект интерактивности, т. е. диалогово- го взаимодействия программы с пользователем. Синтаксис метода confirm():

confirm(*сообщение*).

Если *сообщение* представляет собой вполне определенный набор символов, то его необходимо заключить в кавычки, двойные или одинарные:

confirm("Вы действительно хотите выйти из программы?").

Как и ранее *сообщение* может быть любым: последовательностью символов, заключенной в кавычки, числом (в кавычках или без них), переменной или выражением.

Диалоговое окно, выведенное на экран методом confirm(), можно убрать щелчком на любой из двух кнопок (**OK** или **Отмена**). До тех пор, пока вы не сделаете этого, переход к ранее открытым окнам невозможен. Следовательно, окно, создаваемое посредством confirm(), модальное. Если пользователь щелкнет на кнопке **OK**, то метод вернет логическое значение true (истина, да), а если он щелкнет на кнопке **Отмена** (Cancel), то — false. Возвращаемое значение доступно для дальнейшей обработки в программе и реализации эффекта интерактивности, т. е. диалогового взаимодействия программы с пользователем.

Метод *prompt()*

Метод prompt() позволяет вывести на экран диалоговое окно с сообщением, а также с текстовым полем, в которое пользователь может ввести данные. Кроме того, в окне предусмотрены две кнопки: **OK** и **Отмена** (Cancel). В отличие от alert() и confirm(), данный метод принимает два параметра: сообщение и значение, которое должно появиться в текстовом поле ввода данных по умолчанию. Если пользователь щелкнет на кнопке **OK,** метод вернет содержимое поля ввода данных, а если он щелкнет на кнопке **Отмена**, то возвращается логическое значение false (ложь, нет). По аналогии с предыдущими методами возвращаемое значение можно проанализировать и создать эффект интерактивности.

Синтаксис метода prompt():

prompt(*сообщение, значение_поля_ввода_данн*ых).

Параметры метода prompt()необязательные. Если вы их не укажете, то будет выведено окно без сообщения, а в поле ввода данных Internet Explorer под- ставит значение по умолчанию — undefined (не определено), а другие браузеры — пустую строку. Если вы не хотите, чтобы в поле ввода данных появлялось значение по умолчанию, то в качестве значения второго параметра укажите пустую строку "":

prompt("Введите Ваше имя", "").

Диалоговое окно, выведенное на экран методом prompt(), можно закрыть щелчком на любой из двух кнопок, **OK** или **Отмена**. Пока данное окно открыто, переход к ранее открытым окнам невозможен, т. е. диалоговое окно модальное.

Метод *document.write()*

Метод document.write(список_строк) позволяет вывести в окно браузера список текстовых строк, разделенных запятыми. Если выводимая строка представляет собой (X)HTML-код, то браузер отобразит результат его интерпретации, а не последовательность тегов. Иначе говоря, теги HTML в выводимом сообщении браузер воспринимает как команды, а не просто как текстовые символы.

Метод document.writeln(список_строк)отличается от предыдущего только тем, что выводит указанную строку (или несколько строк), добавляя в конце невидимый управляющий символ перехода на новую строку. Однако этот и другие специальные символы браузер не воспринимает как команды, управляющие отображением, и заменяет их пробелом, т. е. игнорирует. Если требу- ется выполнить какое-либо форматирование выводимого текста, то следует указать соответствующие теги HTML, например, для перевода на новую строку пригодны теги
 и <p>. На рис. 16.2 показан пример вывода в окно браузера трех текстовых строк, первые две из которых содержат HTML-теги.

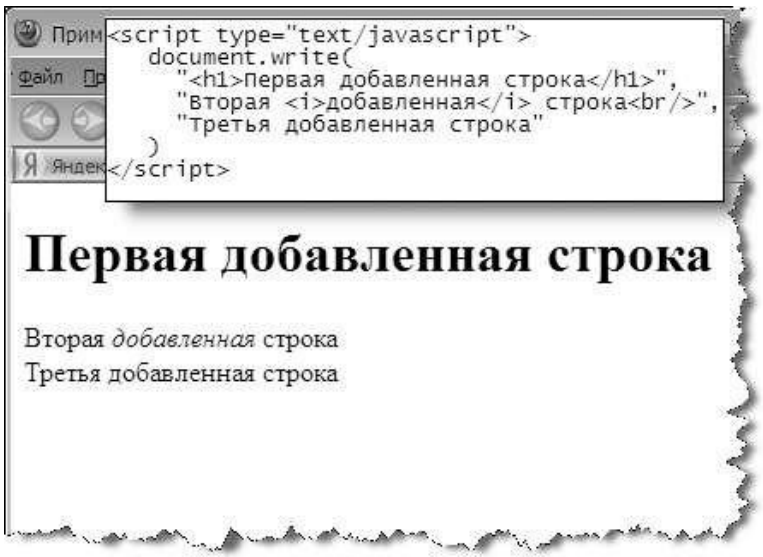

Рис. 2. Пример вывода текстовых строк методом document.write()

Типы данных

Данные в JavaScript, как и в любом другом языке программирования, отно- сятся

к тому или иному типу. Типы данных, в свою очередь, разделяют на примитивные и составные.

Примитивные типы содержат простые значения, с которыми мы встречаемся в повседневной жизни: числа, строки и логические значения вроде "Да" и "Нет". Так, с числами нам приходится оперировать не только при расчетах перед кассовым аппаратом в магазине, но и при решении более сложных ма- тематических задач. Семантику чисел определяет математика. Строки вклю- чают какие угодно символы и их семантика заранее не определяется.

Произвольные данные, без структуры и достаточно четко оформленных идей их дальнейшего употребления, программисту дозволено и рекомендуется оформлять в виде строк. Восприятие строк — дело пользователя, а не интер- претатора языка. Однако применительно к скриптовым (интерпретируемым, а не компилируемым) языкам последнее замечание не вполне точно. Интерпре- тируемые языки, такие как JavaScript и PHP, не "оставляют в покое" строки. Символьные последовательности, участвующие в выражениях с данными других типов, автоматически преобразуются к соответствующему, с точки зрения интерпретатора, типу (см. разд. 16.2.3). Интерпретаторы этих языков пытаются "извлечь" некий смысл, который, возможно, пользователь или про- граммист в них вложил. Например, встречаясь со строкой "25", содержащей число 25, интерпретатор JavaScript попытается привести его к тому или ино- му типу в зависимости от контекста. Для новичков такая поддержка их ин- туиции со стороны интерпретатора языка более чем кстати. Но для искушен- ных программистов, привыкших все держать под своим контролем, это просто "беспредел".

Строка, пусть даже содержащая одни только цифры, не является числом. По- этому арифметические операции над строками, содержащими числа, могут привести к результатам, существенно отличным от тех, которые мы ожидаем в случае действий над самими числами.

Чтобы как-то определиться, мы должны признать, что число — объект, дос- таточно ясно определенный в математике, — в языке программирования представляется символически так или иначе. Строка — это набор произ- вольных символов, которую нельзя интерпретировать, например, как число, дату или что-либо другое. Над строками мы производим одни операции, а над числами — другие. Логическое значение — одно из двух возможных значений, которые обычно обозначаются как true (ИСТИНА) и false (ЛОЖЬ). Для работы с логическими значениями существует своя алгебра,

отличная от арифметики. Мы можем сформулировать некое утверждение, которое в данном контексте может быть истинным или ложным. Для обо- значения оценки истинности логического выражения в языках програм- мирования предусмотрены специальные ключевые слова: true (истина) и false (ложь).

Составные типы данных, в отличие от примитивных, могут содержать разно- родные данные (в том числе и других составных типов). Типичный пример составного типа данных — массив. Массив в JavaScript — это упорядочен- ный набор данных (называемых элементами), каждое из которых принадле- жит к тому или иному типу. Место каждого элемента в массиве фиксировано, а потому доступ к ним возможен по

порядковому номеру (индексу). Объ- ект — еще один составной тип данных, причем настолько общий, что позво- ляет представить все другие составные типы данных, такие как массивы, даты, функции и даже объекты. В объекте данные не обязаны быть упорядо- ченными, как в массиве.

Примитивные типы данных

В JavaScript имеются пять примитивных типов данных (табл. 1).

Таблица 1. *Примитивные типы данных в JavaScript*

Тип данных	**Примеры**	**Описание значений**
Строковый или символьный (String)	"Привет всем!" "т.123-4567" "Сегодня 14.01.2010г."	Последовательность символов, в том числе и пустая, заключенная в кавычки, двойные или одинарные
Числовой (Number)	3.14–567 +2.5 5.7e1667e-28	Десятичное число, представленное в виде последовательности цифр, перед которой может быть указан знак числа (+или –); перед положительными числами не обязательно ставить знак +; целая и дробная части чисел разделяются точкой. Возможно представление в экспоненциальной, шестнадцатеричной и восьмеричной формах. Числа записываются без кавычек. Диапазон чисел: от $\pm 2{,}2250\times10^{-308}$ до $\pm 1{,}7976\times10^{308}$. Диапазон целых чисел: от -2^{31} до $2^{31}-1$
Логический (булевый, Boolean)	true false	Два значения: true(истина, да) и false(ложь, нет)
Пустой (Null)	null	Одно значение — null, указы- вающее на отсутствие какого бы то ни было значения
Неопределенный (Undefined)	undefined	Одно значение — undefined, указывающее, что переменной не

		присвоено никакое значение

Из перечисленных типов только String, Number и Boolean могут содержать настоящие данные, а Null и Undefined служат специальным целям.

Если вы не хотите присваивать переменной какое-либо конкретное значение, то назначьте ей для определенности значение null. Это действие будет отве- чать вашим чаяниям наилучшим образом. Переменная, имеющая значение null, определена. Она имеет пустое значение. Заметим, что пустая строка (не содержащая ни одного символа, даже пробела), а также число 0, — данные, отличающиеся от null.

Если какая-то переменная встречается в выражении впервые, и ей ранее не присвоено никакое значение, то это может вызвать ошибку выполнения, как в листинге 16.1 слева, справа показано, как избежать появления ошибки.

Листинг 1. Пример инициализации переменной

Код с ошибкой

```
<html>
<script type="text/javascript"> alert(x); /* здесь возникнет
                ошибка */
</script>
</html>
```

Код без ошибки

```
<html>
<script type="text/javascript"> x = null;
   alert(x) /* выведет сообщение
        "null", но ошибки не будет */
</script>
</html>
```

Если переменная объявлена с помощью оператора var, например, var x, то она не имеет пока никакого значения и относится к типу undefined (листинг 2).

Листинг 2. Пример переменной типа undefined

```
<html>
<script type="text/javascript">
var x;      // объявление переменной без присвоения ей значения alert(x); //
выведет сообщение "undefined", но ошибки небудет
</script>
</html>
```

Составные типы данных

В JavaScript поддерживаются три составных типа данных: объекты, массивы и

функции. В действительности массивы и функции также являются объектами. Другими словами, такие понятия как массив и функция реализованы в JavaScript в виде объектов.

Объект — понятие, обладающее настолько большой общностью, что противопоставляется исторически более раннему понятию "тип данных". Действительно, объект есть контейнер, содержащий переменные и функции, или одну из этих категорий: только переменные или только определения функций. В свою очередь, переменные внутри объекта могут содержать данные как примитивных, так и составных типов (в том числе и данные типа объект). Последнее обстоятельство как раз и позволяет конструировать сколь угодно сложные типы данных. Если исходить из более традиционного понятия "тип данных", то объект — это сложный (составной) тип данных. С другой стороны, поддержка как примитивных, так и сложно устроенных данных может быть обеспечена соответствующими объектами.

Исторически сложилось, что сначала появилось понятие типа данных (в смысле множества допустимых значений). Более того, в первых языках программирования это было множество однородных значений, например, только чисел, строк или логических величин. Нельзя было сформировать тип данных, состоящий одновременно, например, из чисел и строк (числа и строки — разнородные данные). Далее появились структуры в виде упорядоченного множества (последовательности) данных различных типов. Структура — это составной тип данных, если угодно прибегнуть к такой интерпретации. Наконец, было введено еще более общее понятие объекта, который мог содержать не только разнородные значения (как и структура), но еще и функции (называемые методами). Другими словами, объект может содержать не только пассивные данные (переменные), но и активные (функции).

Обратите внимание, что при разъяснении того, что такое объект, мы использовали понятие "тип данных". Но если мы усвоили понятие "объект", то его, как будто, можно рассматривать в качестве первичного по отношению к понятию "составной тип данных". Однако это можно сделать лишь с некоторой натяжкой. Понятия "тип данных" и "объект" не вполне эквивалентны. Многие типы данных реализуются и представляются в конкретных языках, в том числе и в JavaScript, посредством соответствующих объектов — программных конструкций особого рода. Например, массив реализован и представлен в JavaScript как некий объект под названием Array. Этот объект имеет ряд свойств, содержащих не только элементы представляемого им массива, но и другие важные вещи: количество всех элементов массива и методы работы с ним (например, сортировка элементов, объединение двух массивов). Однако объект — неупорядоченное множество его свойств, а массив — упорядоченное множество его элементов. Кроме того, операции над массивом являются чем-то внешним по отношению к самому массиву — упорядоченному набору данных. По этим и некоторым другим причинам массивы выделяют в особый тип данных, отличающийся от объектов. Как бы то ни было, начинающие, и не только, могут относиться к массивам как к особому типу данных, т. е. как к специальной конструкции, идея и польза которой очевидны. Позже мы уточним это понятие настолько, чтобы его можно было легко применять в программировании.

Аналогично, все, что связано с понятием функции, можно реализовать и представить в виде некоторого объекта. В JavaScript каждая функция реализуется

посредством объекта Function. Такое представление обладает многими достоинствами, которые оказываются востребованными при составлении более или менее сложных программ. Однако любой программист может ограничиться традиционным пониманием функции как блока программного кода, который можно вызвать для исполнения по имени этой функции. В этом смысле функцию можно отнести к особому типу данных. Интерпретаторы, прежде чем начать выполнение кода, анализируют его в целом на присутствие в нем определений функций, т. к. код функций необходимо "усвоить" прежде, чем он будет вызван где-то и когда-то (до или после определения). В отличие от функций, другие операторы выполняются интерпретатором друг за другом в порядке их упоминания в исходном коде.

Если все сказанное внесло некоторую сумятицу в сложившуюся у вас систему понятий, то не расстраивайтесь, со временем все станет на свои места. На первых порах вы можете пропустить как будто путаные общие рассуждения о типах и объектах, сосредоточившись пока на чем-то более понятном (например, на простом определении типа данных). Простейшие типы данных, как я уже говорил, — это примитивные данные, такие как числа, строки символов и двухэлементное множество, содержащее логические значения true и false. Необходимость в других типах данных обусловлена более сложными задачами их обработки.

В JavaScript имеется множество встроенных объектов, обеспечивающих работу с числами, строками, массивами, датами, функциями и др. Некоторые из этих объектов соответствуют, как говорят, типам данных. Например, для работы с массивами имеется объект Array, для манипулирования датами и временем — Date, а для выполнения более или менее сложных математических вычислений — Math. Перечень встроенных объектов приведен в табл. 2, а их подробное описание будет приведено в последующих разделах данной главы.

Таблица 2. *Встроенные объекты в JavaScript*

Объект	**Описание**
Array	Обеспечивает массив данных
Boolean	Соответствует логическому типу данных Boolean
Date	Обеспечивает работу с датами и временем
Error	Обеспечивает хранение информации об ошибках и возможность создания исключительных ситуаций
Function	Обеспечивает возможности, связанные с функциями, например, проверку количества переданных параметров
Global	Обеспечивает встроенные функции для преобразования и сравнения данных
Объект	**Описание**
Math	Обеспечивает множество математических функций и констант. Применяется для выполнения более сложных вычислений, чем те, которые реализуются арифметическими операторами
Number	Соответствует числовому типу данных
Object	Базовый объект, из которого получаются все остальные объекты, в

	том числе и пользовательские
RegExp	Обеспечивает сложные способы обработки строк на основе так называемых регулярных выражений
String	Соответствует строковому типу данных

Автоматическое преобразование типов данных

При составлении выражений на языке программирования нередко бывает, что вовлекаемые в обработку данные принадлежат различным типам, в то время как операции над ними могут быть корректно выполнены, если данные однотипны. Например, пусть переменная *x* имеет числовое значение 2, а переменная *y* — строковое значение "3". С обывательской точки зрения и то, и другое — числа, но для интерпретатора языка JavaScript это разные вещи: переменная *x* содержит само число, а переменная *y* — строку, содержащую число. Числа обычно составляются из цифр, знаков положительного и отрицательного числа, разделителя целой и дробной частей, а также других специальных символов. Строка — набор произвольных символов, который заключают в кавычки. Что будет означать операция сложения этих разнотипных данных, суммирование чисел или склейку соответствующих строк? Попытка сложить значения этих переменных в JavaScript даст в результате не число 5, а строку "23" (рис. 3).

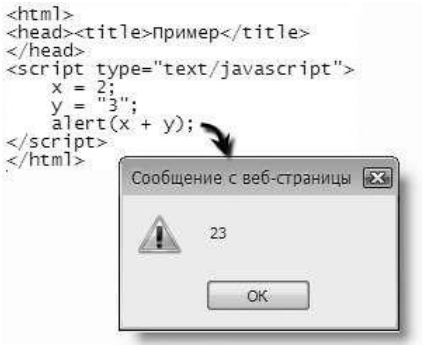

Рис. 3. Сложение числа и строки дает в результате строку

В данном случае интерпретатор JavaScript автоматически преобразует число 2 в строку "2", а операцию "+" выполнит как склейку строк "2" и "3".

JavaScript — язык со слабым или, как еще говорят, динамическим контролем типов данных. Это означает сле- дующее:

- Одна и та же переменная может принимать значения различных типов. Переменная создается при первом ее упоминании в программе, а тип дан- ных, которые ей можно присвоить, не задан раз и навсегда. В одном месте программы этой переменной можно присвоить значение одного типа, а в другом месте — иного типа. Например, вполне допустима такая после- довательность операторов присваивания: x=2; x="Вася".

- При выполнении операций над данными различных типов интерпретатор автоматически приводит их к некоторому общему требуемому типу. Например, при сложении числа и строки число приводится к строковому типу, в результате вторая строка приписывается ("приклеивается") к пер- вой. Так, результат выражения 125 +

"Вася" — строка "125Вася". Рассмот- рим еще один пример. Иногда в выражении по смыслу требуется логиче- ский тип. Так, в операторе условия if(x){…} (читается "если истинно x, то выполнить {…}") подразумевается, что переменная x должна иметь логиче- ское значение (true или false). Однако в действительности она может принимать значение любого типа, но интерпретатор приведет его к логическому типу автоматически сообразно с некоторыми правилами.

Как происходит автоматическое преобразование типов, очень важно знать, чтобы избежать возможных недоразумений. Однако существуют методы принудительного приведения к заданному типу, которые позволяют контро- лировать типы данных вручную.

Преобразование строк (*String*)

Правила автоматического преобразования строк (данных типа String) в данные других типов:

- **Преобразование в Boolean** — в результате получается false (ложь), если строка пуста (""), т. е. не содержит ни одного символа, даже пробела (строка, содержащая один или более пробелов не пуста); в противном случае — true(истина).
- **Преобразование в Number** — происходит анализ строки как числового литерала с целью получения подходящего значения. Если этот процесс заканчивается неудачей, то возвращается константа NaN (от Not a Namber — не число); в противном случае возвращается число. Например, строка "5.2" преобразуется в число 5.2, а в результате преобразования строки "100px" будет получено NaN.
- **Преобразование в Object** — возвращается объект String, свойство value которого равно значению данной строки.

Преобразование чисел (*Number*)

Правила автоматического преобразования чисел (данных типа Number) в данные других типов:

- **Преобразование в Boolean —** в результате получается false (ложь), если число равно нулю или NaN (Not a Number — не число); в противном случае — true (истина). NaN— константа, содержащаяся в объекте Number.

- **Преобразование в String** — в результате получается строка, содержащая (представляющая) данное число. Например, число 3.14 преобразуется в строку "3.14".
- **Преобразование в Object** — возвращается объект Number, свойство value которого равно значению данного числа.

Преобразование логических значений (*Boolean*)

Правила автоматического преобразования логических значений (данных типа Boolean) в данные других типов:

- **Преобразование в Number** — в результате получается 1, если true (истина); в противном случае — 0.
- **Преобразование в String** — в результате получается строка "true", если true; в противном случае — строка "false".
- **Преобразование в Object** — возвращается объект Boolean, свойство valueкоторого равно данному логическому значению.

Преобразование пустого значения (*null*)

Правила автоматического преобразования пустого значения (данных типа null) в данные других типов:

- **Преобразование в Boolean —** в результате получается false(ложь).
- **Преобразование в Number** — в результате получается 0.
- **Преобразование в String —** в результате получится строка "null".
- **Преобразование в Object** — генерируется исключительная ситуация TypeError (ошибка преобразования типа).

Преобразование неопределенного значения (*undefined*)

Правила автоматического преобразования неопределенного значения (дан- ных типа undefined) в данные других типов:

- **Преобразование в Boolean —** в результате получается false(ложь).
- **Преобразование в Number** — в результате получается NaN(не число).
- **Преобразование в String —** результате получится строка "undefined" (не определено).
- **Преобразование в Object** — генерируется исключительная ситуация TypeError (ошибка преобразования типа).

Принудительное преобразование типов данных

При попытке выполнить операции над разнотипными данными последние автоматически приводятся к определенному типу так, чтобы выражение могло быть выполнено. Во многих случаях результат оказывается ожидаемым, но иногда возникают недоразумения, устранить которые часто удается принудительным приведением данных к требуемому типу. Например, пусть переменная width="100px" хранит значение ширины окна в пикселах, причем единица измерения (px — пикселы) указана в самом этом значении строкового типа. Если мы захотим увеличить размер окна на 10 пикселов, используя для этого выражение width = width + 10, то значение переменной width станет равным "100px10", а не ожидаемой величине 110. В данном случае интерпретатор автоматически приведет число 10 к строко- вому значению "10" и припишет его справа к строковому значению "100px". Чтобы добиться своего, нам потребуется принудительно преобразовать строку "100px" в соответствующее число, а именно в 100.

Для преобразования строк в числа в JavaScript предусмотрены встроенные

функции parseInt() и parseFloat(). В действительности это методы объекта global, предназначенного для хранения общедоступных данных и методов. В силу глобальности объекта global для обращения к его методам не нужно указывать сам объект, т. е. для вызова parseInt(x) достаточно просто записать parseInt(x). Возможно, по этой причине методы объекта global называют встроенными функциями JavaScript.

Функция parseInt(*строка*, *основание*)

преобразует указанную в параметре строку в целое десятичное число. Второй параметр, *основание*, определяет систему счисления, в которой представлено содержащееся в строке число (8, 10 или 16); если основание не указано, то предполагается 10, т. е. десятичная система счисления. При этом число ок- ругляется простым отбрасыванием дробной части.

Примеры

```
parseInt("3.14")      // результат = 3
parseInt("-7.875")    // результат = -7
parseInt("123")       // результат = 123
parseInt("Саша")      // результат =NaN, т. е. не является числом parseInt("25
руб. 50 коп.") // результат =25
parseInt("25.5e3")    // результат = 25
parseInt("15",8)      // результат = 13
parseInt("0xFF",16)// результат = 255 parseInt("ff",16)
                      // результат = 255
```

Функция parseFloat(*строка*) преобразует указанную строку в десятичное число с плавающей разделительной (десятичной) точкой.

Примеры

```
parseFloat("3.14")    // результат = 3.14
parseFloat("-7.875")  // результат = -7.875  parseFloat
("123")               // результат = 123
parseFloat ("Саша") // результат = NaN, т. е. не является числом
parseFloat("25 руб. 50 коп.") // результат =25
parseFloat ("25.5") // результат = 25.5 parseFloat ("25.5
рублей") // результат = 25.5 parseFloat("25.5e3") // результат
= 25500
```

Данные функции исследуют строку на содержание в ней числа. Если первый символ не цифра или знак числа, то сразу возвращается NaN. В противном случае анализируется следующий символ строки и если это не цифра, то ана- лиз прекращается и возвращается число, выделенное на предыдущих этапах. Метод parseInt() закончит работу, как только встретит символ, не являющий- ся цифрой (например, точку, пробел или букву). Поэтому строка, представ- ляющая собой число в

экспоненциальной форме, будет преобразована в число, содержащее только целую часть мантиссы. Например, parseInt("25.5e3") вернет 25, а не 25500. Метод parseFloat() анализирует строку глубже: деся- тичная точка и даже буква "e", строчная или прописная, а также знак и цифры за ней будут вовлечены в обработку для получения числа.

Числа в строки можно преобразовать еще проще. Для этого достаточно к пустой строке прибавить это число, т. е. воспользоваться оператором сло- жения +. Например, вычисление выражения "" + 3.14 даст в результате строку "3.14". Понятно, что в данном случае преобразование выполняет сам интерпретатор, используя свой "интеллект".

Для определения того, является ли значение выражения числом, служит ме- тод (или встроенная функция)

isNaN(*значение*),

который возвращает результат логического типа. Если указанное значение не является числом, функция возвращает true, иначе — false. Однако здесь термин "число" не совпадает с понятием "значение числового типа". Функция

isNaN() считает числом и данные числового типа, и строку, содержащую только число. Логические значения также идентифицируются как числа. При этом значению true соответствует 1, а значению false — 0. Таким образом, если isNaN() возвращает false, то это указывает, что значение параметра имеет числовой тип, либо является числом, преобразованным в строковый тип, либо является логическим (true или false).

Примеры

```
isNaN(1234) // результат false (т. е. это число)
isNaN("1234") /* результат false (т. е. эточисло,
                    хотя и в виде строки)*/
isNaN("25 рублей")// результат true (т. е. это не число)
isNaN("Саша") // результат true (т. е. это не число) isNaN(true) //
результат false
isNaN(false)  // результат false
```

Как видно, значение isNaN(x) для любого логического значения x возвраща- ет false, что указывает на то, что x — число. Дело в том, что функция isNaN пытается привести значение x к числовому типу по правилам автоматическо- го преобразования типов (см. разд. 16.2.3). Логические значения true и false преобразуются по этим правилам в числа 1 и 0 соответственно. Автоматиче- ское приведение значений "Саша" и "25 рублей" к числовому типу заканчи- вается получением значения NaN и поэтому функция isNaN() возвращает для этих значений true.

Кроме рассмотренных, в JavaScript имеются и другие функции приведения данных к заданному типу:

- Number(*выражение*) — десятичное число или NaN (если не удалось преоб- разовать указанное *выражение* в число). Эта функция работает следующим образом:

- если *выражение* есть число, то возвращается это же число;
- если *выражение* есть логическая величина, то возвращается 1 или 0 в зависимости от ее значения (если true, то 1; если false, то 0);
- если *выражение* есть строка, то функция пытается преобразовать ее в число как описанные ранее функции parseInt()и parseFloat();
- если *выражение* есть udefined(не определено), то результатом является NaN(не число).

- String(*выражение*) — приводит данные, получающиеся в результате вычисления выражения, которое указано в качестве параметра, к строковому типу. Преобразование происходит по следующим правилам:
  - если значение выражения имеет строковый тип, то возвращается это же значение;
  - если значение выражения имеет числовой тип, то возвращается строка, содержащая это число;
  - если значение выражения имеет логический тип, то возвращается стро- ка "true"или "false"в зависимости от значения выражения;
  - если значение выражения не определено (undefined), то возвращается пустая строка "".
- Boolean(*выражение*) — приводит данные, получающиеся в результате вычисления выражения, которое указано в качестве параметра, к логическо- му типу. Преобразование происходит по следующим правилам:
  - если значение выражения имеет логический тип, то возвращается это же значение;
  - если значение выражения имеет числовой тип, то возвращается true, если число не равно нулю, и false— в противном случае;
  - если значение выражения имеет строковый тип, то возвращается true, если строка не пуста, и false — в противном случае; напомню, что строка, содержащая только пробелы, не пуста.
- Array(*элемент0* ,*элемент1*, ..., *элементN*) — создает массив из эле- ментов, указанных в качестве параметров.

В действительности перечисленные функции являются обращениями к соответствующим объектам: Number, String, Boolean и Array. Например, String(*выражение*) — обращение к так называемому статическому строково- му объекту. Об этом и о других объектах будет рассказано далее.

Переменные и оператор присваивания

Имена переменных

Как уже упоминалось, переменная — это контейнер для хранения значений. Значения могут быть любых типов, предусмотренных в JavaScript, а присваи- вание происходит с помощью оператора, обозначаемого знаком равенства =. Например в выражении x = 5 имя (идентификатор) переменной — x, а зна- чение, которое ей присваивается (приписывается, назначается) — 5(в данном

случае это числовое значение), следовательно, переменная x числовая. Если мы теперь запишем в своей программе выражение x = "Вася", то переменная x приобретет новое значение строкового типа "Вася".

Переменная имеет имя — последовательность букв, цифр, символа подчеркивания и даже символа $, которая не должна начинаться с цифры.

Примеры правильных имен переменных:

myname, myName, _myname, my_Name , myName134,

$myname. Примеры неправильных имен переменных:

my name, 310group, 1000 $.

Имя, соответствующее указанным правилам, может быть каким угодно. Од- нако желательно, чтобы оно отражало суть содержащихся в соответствующей переменной значений и/или цель ее использования. Если в имени переменной вы хотите указать несколько слов, то разделите их символом подчеркивания или напишите каждое из этих слов (начиная со второго) с прописной буквы, например, my_first_name, myFirstName. Нередко первый символ в имени переменной указывает на ее тип. Например, в имени sMyName первый символ обозначает, что данная переменная содержит данные строко- вого типа (string). Вы можете придумать свой стиль образования имен пе- ременных. Важно лишь то, чтобы он был понятен хотя бы вам самому. Тем не менее, профессиональные программисты JavaScript не рекомендуют выби- рать в качестве первого символа имени переменной подчеркивание или знак

$.

JavaScript регистрозависимый язык. Это означает, что изменение регистра символов (с прописных на строчные и наоборот) в имени переменной приво- дит к другой переменной. Например, myvar, Myvar и MYVAR — различные пе- ременные.

При выборе имен переменных (а также имен функций) недопустимы зарезервированные ключевые слова, которые используются или, планируются к применению в последующих версиях в качестве элементов синтаксиса. Так, не следует выбирать в качестве имен слова из списка:

abstract	else	int	super
boolean	extends	interface	switch
break	false	long	synchronized
byte	final	native	this
case	finally	new	throw
catch	float	null	throws
char	for	package	transient

class	function	private	true
const	goto	protected	try
continue	if	public	typeof

default	Implements	reset	var
delete	import	return	void
do	in	short	while
double	instanceof	static	with

Создание переменных

Здесь мы затронем чрезвычайно важную тему видимости переменных. Она актуальна не только в JavaScript, но и вообще во всех языках программирова- ния. Интересно, что писать вполне работоспособные скрипты для сайтов уда- ется в большинстве случаев и тем, кто данной темой совсем не владеет. Од- нако это "пиррова победа", обусловленная благополучным стечением обстоятельств или простотой задачи. Стоит лишь немного усложнить сцена- рий, потребовать взаимодействия нескольких программных блоков, как сразу же возникнут недоразумения и ошибки, обусловленные, скорее всего, недо- пониманием того факта, что переменная может быть, а может и не быть дос- тупной в том или ином блоке кода или контексте выполнения сценария.

Переменную в JavaScript можно создать с помощью оператора присваивания:

имя_переменной = *значение*;

Например, в выражении myname = "Вадим" переменной с именем (идентифи- катором) myname присваивается значение строкового типа "Вадим". Пока дан- ная переменная не получит какого-нибудь значения другого типа, она оста- нется строковой.

Кроме того, для объявления переменной предусмотрен специальный опера- тор var, например,

var myname; myname = "Вадим"; или еще короче:

var myname = "Вадим";

В данном примере одновременно с объявлением переменной ей присваивает- ся значение. Присваивать значения не обязательно всем объявляемым пере- менным. Например, в выражении

var x, y, z = "Привет!"

объявлены (созданы) переменные x, y и z, причем только переменной z при- своено конкретное значение. Переменные x и y, которым пока не присвоены

значения в программе, относятся к типу undefined и имеют такое же значение. На рис. 4 показан соответствующий пример.

Обратите внимание, что с помощью одного ключевого слова (оператора) var можно создать (инициализировать) сразу несколько переменных. При этом имена переменных и/или операторы присваивания разделяют запятыми.

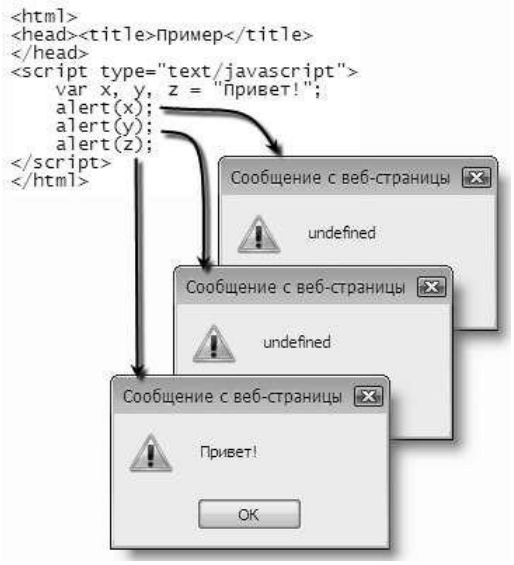

Рис. 4. Переменные x и y объявлены, но не определены, а переменная z определена

Хотя переменную в JavaScript можно создать как с указанием оператора var, так и без него, просто с помощью оператора присваивания, хороший стиль программирования состоит в применении var. Он хорош потому, что позволяет читателю программного кода достаточно четко различить, где переменная была объявлена впервые, а где она просто используется. Кроме того, оператор var позволяет задать так называемые локальные переменные, которые доступны только из некоторой части программного кода или, как еще говорят, в рамках некоторого контекста. И это, пожалуй, самое главное назначение данного оператора. Дело в том, что переменные могут быть глобальными и локальными. Начинающие программисты обычно не чувствуют разницу, пока не встретятся с недоразумениями на практике.

Переменные, введенные в обиход с помощью оператора присваивания (без оператора var) глобальные, т. е. видимы или, иначе говоря, доступны во всей программе, включая и программные блоки (например, тела функций), если

в них не приняты специальные меры, о которых будет рассказано далее. Вместе с тем, глобальные переменные можно создать и с помощью оператора var.

Итак, мы приступаем к подробному рассмотрению очень важной темы — области видимости и контекста переменной. Область видимости — это часть программы, в которой переменная доступна или, как еще говорят, видна. Видимая переменная доступна в программе, а невидимая — нет. Контекст — набор определенных данных, образующих среду выполнения программы. При запуске браузера (интерпретатора JavaScript) создается глобальный кон- текст, который помимо специфических объектов браузера и загруженного в него документа содержит объекты собственно JavaScript (например, объекты массива Array, даты Date, объект для сложных математических вычислений Math и др.). При активизации обработчика события (например, после щелчка кнопкой мыши на элементе Web-страницы) создается локальный контекст этого обработчика. То же самое происходит и при вызове функции: интерпретатор создает локальный контекст, существующий до тех пор, пока не завершится выполнение этой функции. Все переменные, объявленные в теле функции с помощью оператора var, существуют только в рамках контекста этой функции во время ее выполнения. По завершении работы функции этот

контекст уничтожается. Контексты могут быть иерархически вложены друг в друга. Интерпретатор, встречая переменную в некотором контексте, анализирует, определена ли она в нем в нем. Если да, то это — локальная переменная, существующая в данном контексте. В противном случае интерпретатор ищет определение переменной в объемлющем контексте. Эти действия выполняются рекурсивно, пока не будет найден контекст, содержащий определение переменной. Не исключено, что при выходе на глобальный контекст определение переменной так и не будет найдено. Тогда переменная оказывается неопределенной.

В ранее рассмотренных примерах все переменные были глобальными. В следующем примере (рис. 5) переменная x глобальная и имеет значение "Привет!", несмотря на то, что после первичного ее объявления и присваивания значения эта же переменная вновь объявляется, но без присваивания.

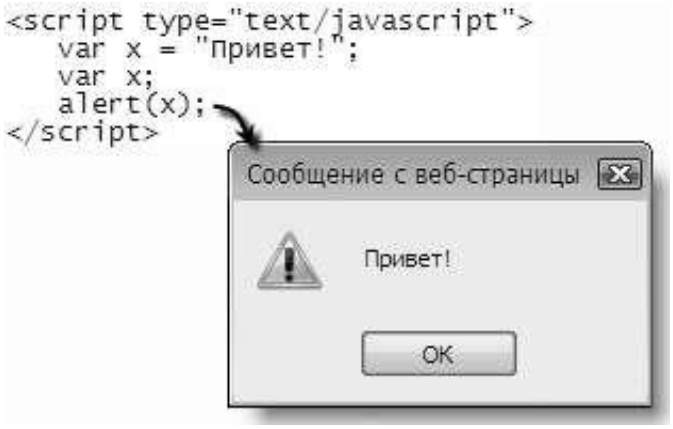

Рис. 5. Переменная хглобальная

Здесь интерпретатор, в контексте метода alert() не находит определения переменной x, а в объемлющем контексте он не останавливается на промежуточном определении var x, поскольку в данном пункте x остается undefined. Продолжение поиска в рассмотренном примере заканчивается успехом: выражение var x = "Привет!" полностью определяет переменную x. Это значение метод alert() выводит в диалоговом окне.

В теле функции можно использовать переменные, объявленные в программном коде более высокого уровня, т. е. в коде, объемлющем определение данной функции. Например, в основной программе, из которой данная функция вызывается. Будут ли такие переменные видны в теле функции? А если в теле функции изменить значения таких переменных, то станут ли они доступными во внешней программе?

На рис. 6 представлен исходный код сценария, иллюстрирующий видимость переменной в контексте функции myfunc() и в глобальном контексте, а также результат его выполнения в окне браузера. Переменная x определена в основной программе (глобальном контексте) вне тела функции и является глобальной. Она видна также и в теле функции (в контексте ее выполнения), а новое значение "До свидания!" этой переменной, присвоенное в теле функции, остается таким же и после завершения работы этой функции, т. е. в глобальном контексте.

Теперь немного изменим сценарий: переменную x в теле функции объявим с помощью оператора var. Код этого сценария в текстовом редакторе и результат его выполнения в браузере показаны на рис.7. В данном случае при первом упоминании в

теле функции myfunc() переменная x оказывается не- определенной (undefined). Действительно, при вызове функции интерпретатор, встречая в первый раз переменную x, ищет ее определение в локальном контексте (т. е. в теле этой функции) и находит его: var x = "До свидания!".

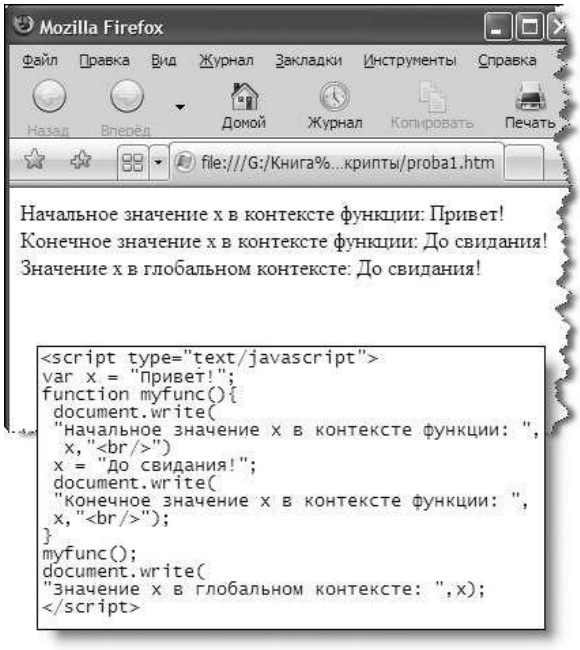

Рис. 6. Глобальная переменная x, видимая в теле функции

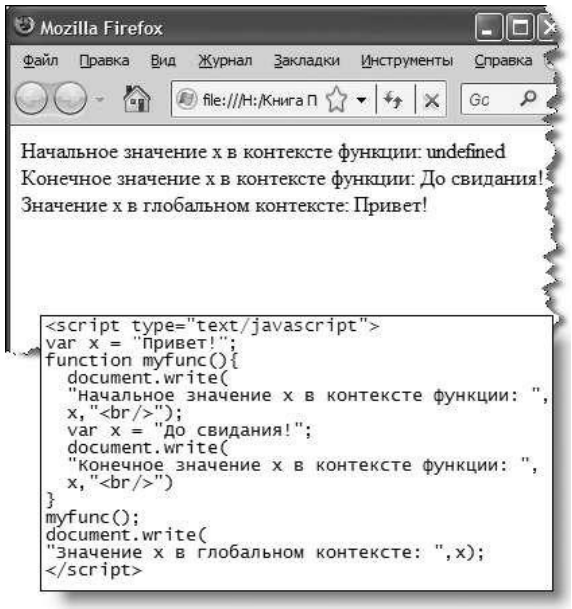

Рис. 7. Локальная переменная xв теле функции

Следовательно, это локальная переменная. Однако определение переменной x в теле функции следует после ее первого упоминания в пределах кода той же функции. Поэтому при выполнении выражения

document.write("Начальное значение x в контексте функции: ", x,"
") она еще не определена, и потому имеет значение undefined. Зато в выражении document.write("Конечное значение x в контексте функции: ",x,"
"),

следующем за определением var x = "До свидания!", она имеет заданное в этом определении значение. Но значение "До свидания!" переменная x имеет только при выполнении функции myfunc(). Вне этого контекста (т. е. после завершения работы функции) переменная x видна как имеющая значение "Привет!".

Таким образом, указание оператора var перед именем переменной в теле функции делает ее локальной, отличной от одноименной глобальной переменной. При этом говорят, что оператор var в программном блоке (например, в теле функции) маскирует эту переменную, т. е. делает невидимой в объемлющем контексте. На рис. 8 приведен еще один пример, показывающий маскирующий эффект оператора var в теле функции.

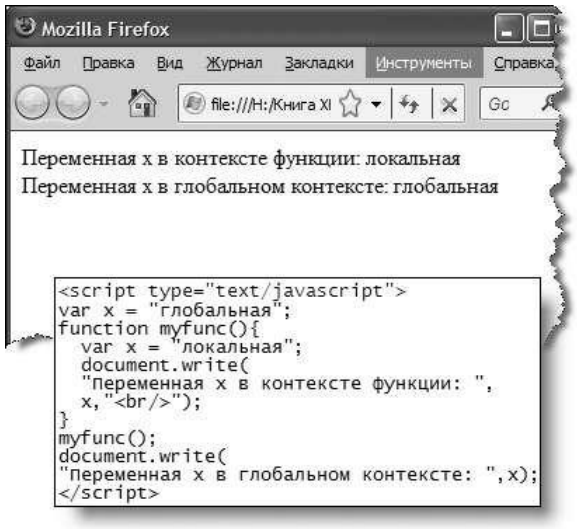

Рис. 8. Маскирующий эффект оператора var