Федеральное агентство по образованию
Волгоградский государственный педагогический университет

А.Н.Сергеев

JavaScript

основы построения, базовые конструкции

Дидактический практикум

Волгоград
2004

Рецензенты:

Жданович П.Б., кандидат физико-математических наук.

Сергеев А.Н.

JavaScript: основы построения, базовые конструкции. Дидактический практикум. – Волгоград, 2004.

Дидактический практикум «JavaScript: основы построения, базовые конструкции» рассчитан на тех, кто впервые изучает возможности программирования в Интернет. Пособие построено на основе практического введения в JavaScript, что позволяет осваивать язык поэтапно, получая навык практического программирования с самого первого занятия. Рекомендуется для студентов педагогических ВУЗов и школьников, изучающих технологии создания ресурсов Интернет.

Содержание

Жданович П.Б., кандидат физико-математических наук.....2

Содержание.....3

Введение.....4

Занятие 1. Практическое введение в JavaScript.....6

Вариация первая: самая простая.....6
Вариация вторая: с подгружаемым исходным текстом.....8
Вариация третья: с переменной и функциями.....9
Вариация четвертая: с диалоговой панелью сообщения.....11
Вариация пятая: с диалоговой панелью ввода информации.....12
Вариация шестая: обработка события.....13
Задания для самостоятельного выполнения.....15

Занятие 2. Основы построения языка JavaScript.....16

Переменные в JavaScript.....16
Объявление переменных.....16
Присвоение значения переменным.....17
Типы данных в JavaScript.....18
Преобразование типов данных.....20
Операторы языка JavaScript.....21
Функции в языке JavaScript.....25

Занятие 3. Базовые алгоритмические конструкции.....25

Организация условных переходов.....25
Задания для самостоятельного выполнения.....27
Операторы цикла 28
Задания для самостоятельного выполнения.....31

Занятие 4. Встроенные объекты JavaScript.....31

Массивы в JavaScript.....31
Задания для самостоятельного выполнения.....33
Работа со строками 33
Задания для самостоятельного выполнения.....37
Математические вычисления в JavaScript.....37
Задания для самостоятельного выполнения.....41

Литература.....43

Введение

Язык сценариев JavaScript предназначен для создания интерактивных HTML-документов. С помощью сценариев поддерживается диалог с пользователем, создаются различные визуальные эффекты, осуществляется навигация по страницам сайта, обрабатываются данные, представленные с помощью различных элементов управления и многое другое.

Язык JavaScript не предназначен для создания автономных программ или аплетов, и в этом он сильно отличается от других языков, таких как C, Pascal или Java. Конструкции JavaScript встраиваются непосредственно в исходный текст документов HTML и обрабатываются браузером по мере загрузки этих документов.

JavaScript является интерпретируемым языком. Категорию языков, к которой относится JavaScript, в литературе называют языками сценариев (скриптов). Основное отличие сценария от традиционной программы – в коротком жизненном цикле его выполнения. Сценарии, как правило, создаются для того, чтобы запуститься, выполнить некоторое действие и прекратить свою работу.

Еще одна важная особенность языка JavaScript - это его объектная ориентированность. Программистам доступны многочисленные объекты, как самого языка, так и внешние объекты браузера и загруженного в него документа HTML

Не менее важно, что с помощью сценариев JavaScript, встроенных в документы HTML, можно обрабатывать события. Эти события возникают в результате выполнения пользователем различных операций над документом HTML, загруженным в окно браузера, что позволяет создавать интерактивные документы, способные изменяться в зависимости от действий пользователя.

Язык JavaScript несложен в изучении. Все, что вам потребуется для начала работы с ним – это браузер и простой текстовый редактор. Вы сможете создавать сложные сценарии и выполнять их прямо на своих компьютерах, без привлечения каких-либо серверных Web-расширений. Вместе с тем, следует отметить, что разные браузеры могут по-разному интерпретировать сценарии JavaScript, поэтому создавая «серьезные» сценарии, предназначенные для публикации в Интернет, не стоит ограничиваться отладкой сценария лишь в одном браузере, под управлением одной операционной системы. Желательно проводить проверку на разных компьютерах и обязательно стараться придерживаться документированных правил написания программ на JavaScript.

Настоящее пособие предназначено для тех, кто впервые изучает JavaScript и технологии программирования для Интернет. Пособие построено на основе практического введения в новый язык программирования. Это

означает, что вы с самого первого занятия сможете создать несложные, но уже вполне состоявшиеся сценарии JavaScript.

Второе занятие посвящено основам построения языка JavaScript. Здесь рассматриваются вопросы объявления переменных, типов данных, использования операторов, создания подпрограмм.

Третье занятие посвящено организации в сценариях JavaScript базовых алгоритмических конструкций – условных переходов и циклов.

Завершающее, четвертое занятие посвящено встроенным объектам JavaScript, которые позволяют выполнять такие важные операции любого языка программирования, как использование массивов, обработка строк и выполнение математических операций.

На каждом занятии, предполагающем практическую работу, вам будут предлагаться задания для самостоятельного выполнения. Цель этих заданий – не только получение практического опыта в написании сценариев на JavaScript. Задания подобраны так, чтобы вы смогли обратить внимание на отличительные особенности языка JavaScript (по сравнению с другими языками программирования) и технологии создания программ в среде Интернет.

Занятие 1. Практическое введение в JavaScript

Изучение нового языка программирования начнем с простого примера – классической программы "Hello, world!". Эта программа впервые была составлена создателями языка C и выводила на экран одну лишь фразу: “Hello, World!”. Мы рассмотрим несколько вариааций такой программы, демонстрирующих различные возможности JavaScript.

Вариация первая: самая простая

На первом этапе составим сценарий JavaScript, который вставляет слова "Hello, world!" непосредственно в документ HTML. Исходный текст документа с подобным сценарием представлен в примере 1.1.

Пример 1.1

```
<HTML>
  <HEAD>
    <TITLE>Hello, world!</TITLE>
  </HEAD>
  <BODY>
    <H1>JavaScript Test</H1>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
      document.write("Hello, world!");
    // -->
    </SCRIPT>
  </BODY>
</HTML>
```

Рассмотренный в примере документ, как того требуют правила языка HTML, ограничен тегами `<HTML>`, `</HTML>` и состоит из двух разделов. Раздел заголовка выделяется тегами `<HEAD>` и `</HEAD>`, а раздел тела документа – тегами `<BODY>` и `</BODY>`. Программа JavaScript этом примере встроена в тело документа HTML при помощи тегов `<SCRIPT>` и `</SCRIPT>`, как это показано ниже:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  document.write("Hello, world!");
// -->
</SCRIPT>
```

С помощью тега `<SCRIPT>` можно встроить в документ сценарий, составленный на языке JavaScript или VBScript. Язык указывается с помощью параметра `LANGUAGE`. Текст сценария оформлен как комментарий с

применением тегов <!-- и -->. Это сделано для того, чтобы сценарий не вызывал проблем у пользователей, браузеры которых не могут работать с JavaScript. Такие браузеры просто проигнорируют сценарий. Обратите внимание на строку, которой завершается комментарий: // --> Перед символами --> записаны два символа /. Это позволяет обеспечить работоспособность сценария в различных браузерах. Некоторые из них (например, Netscape Navigator) в сценариях JavaScript рассматривают строку --> как ошибочную. Символы // используются в JavaScript для выделения комментариев и предписывают браузерам игнорировать символы, записанные после них (в том числе и -->). Для обозначения комментариев можно использовать также конструкцию /*...*/. Этот способ удобен, если комментарий содержит несколько строк.

Наша первая программа весьма проста и содержит только одну строку:

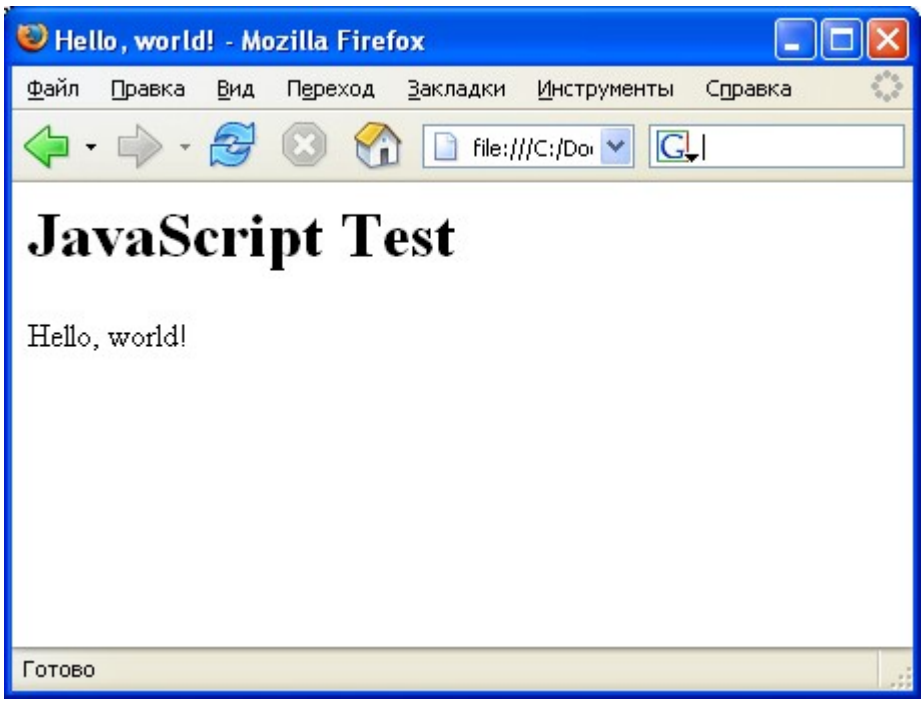
```
document.write("Hello, world!");
```

Здесь для объекта с именем document вызывается метод write. В качестве параметра ему передается текстовая строка "Hello, world!". Строка закрывается символом точка с запятой, который служит разделителем операторов. Объект document - это документ HTML, загруженный в окно браузера. Он содержит в себе объекты, свойства и методы, предназначенные для работы с элементами этого документа HTML, а также для взаимодействия с другими объектами. Метод write записывает в тело документа HTML приветственную строку "Hello, world!". При этом документ будет выглядеть так, как будто эта строка находится в нем на месте сценария:

```
<HTML>
  <HEAD>
    <TITLE>Hello, world!</TITLE>
  </HEAD>
  <BODY>
    <H1>JavaScript Test</H1>
    Hello, world!
  </BODY>
</HTML>
```

На рисунке 1.1 приведен пример работы рассмотренного сценария.

Рисунок 1.1. Пример работы простейшего сценария



Вариация вторая: с подгружаемым исходным текстом

Исходный текст любого сценария должен включаться в документы HTML. Однако, есть техническая возможность оформлять программы на JavaScript в отдельных файлах, а в страницах HTML указывать на эти файлы ссылки. Браузер, загружая оформленные подобным образом HTML документы, загружает оформленные в отдельных файлах сценарии и подставляет их вместо соответствующих ссылок. Такой способ включения JavaScript сценариев удобен, если один и тот же сценарий должен быть включен во множество документов HTML, или же если есть необходимость скрыть исходный код от просмотра пользователями (через просмотр источника).

Ссылки на файлы с подгружаемыми скриптами оформляются с помощью параметра `SRC` тега `<SCRIPT>`, допускающего указывать адрес URL файла сценария. Следующий пример демонстрирует использование параметра `SRC`. В примере 1.2 находится исходный текст документа HTML, содержащий ссылку на файл сценария `hello.js`.

Пример 1.2

```
<HTML>
  <HEAD>
    <TITLE>Hello, world!</TITLE>
  </HEAD>
  <BODY>
    <H1>JavaScript Test</H1>
    <SCRIPT LANGUAGE="JavaScript" SRC="hello.js">
    </SCRIPT>
  </BODY>
```

```
</HTML>
```

Ссылка оформлена с применением тегов <SCRIPT> и </SCRIPT>, однако между этими тегами нет ни одной строчки исходного текста. Этот текст перенесен в файл hello.js (пример 1.3).

Пример 1.3. Файл hello.js

```
document.write("<HR>");
document.write("Hello, world!");
document.write("<HR>");
```

Результат работы сценария, оформленного подобным образом, будет полностью идентичен рассмотренному ранее примеру (рисунок 1.1), за исключением горизонтальных линий, выделяющих приветственную фразу “Hello, world”. Эти линии создаются с помощью тегов <HR>, вставленных в тело документа HTML сценарием JavaScript.

В параметре SRC рассмотренного примера задано только имя файла, так как он находится в том же каталоге, что и ссылающийся на него файл документа HTML. Однако можно указать и относительный путь, и полный адрес URL, например:

```
<SCRIPT LANGUAGE="JavaScript"
SRC="http://www.myserver.ru/scripts/hello.js">
```

Существенно, чтобы файл, в котором находится исходный текст сценария JavaScript, имел тип js. В противном случае сценарий работать не будет.

Вариация третья: с переменной и функциями

В сценариях JavaScript активно применяют функции и переменные. Приведем исходный текст простой программы, выводящей фразу “Hello, World!” (рисунок 1.1), в которой, однако, используется переменная и функция (пример 1.4).

Пример 1.4.

```
<HTML>
  <HEAD>
    <TITLE>Hello, world!</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    var szHelloMsg = "Hello, world!";
    function printHello()
    {
      document.write(szHelloMsg);
    }
    // -->
    </SCRIPT>
```

```
  </HEAD>
  <BODY>
    <H1>JavaScript Test</H1>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    printHello();
    // -->
    </SCRIPT>
  </BODY>
</HTML>
```

Прежде всего, обратите внимание на область заголовка документа, выделенную тегами `<HEAD>` и `</HEAD>`. В этой области расположено определение переменной и функции, оформленное с применением тегов `<SCRIPT>` и `</SCRIPT>`:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  var szHelloMsg = "Hello, world!";
  function printHello()
  {
    document.write(szHelloMsg);
  }
// -->
</SCRIPT>
```

Кроме того, в теле документа HTML есть еще один раздел сценариев, выделенный аналогичным образом:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  printHello();
// -->
</SCRIPT>
```

Переменная с именем `szHelloMsg` определяется при помощи оператора `var`, причем ей сразу же присваивается начальное значение - текстовая строка `"Hello, world!"`.

Язык JavaScript не является типизированным. Это означает, что программист не может указать явным образом тип создаваемых им переменных. Этот тип определяется интерпретатором JavaScript автоматически, когда переменной в первый раз присваивается какое-либо значение. В дальнейшем можно легко изменить тип переменной, просто присвоив ей значение другого типа. Отсутствие строгой типизации упрощает создание сценариев, особенно для непрофессиональных программистов, однако может привести к

ошибкам. Поэтому необходимо внимательно следить за тем, какие типы данных применяются. Этому способствует использование префиксов имен, по которым можно судить о типе переменной. Например, текстовые строки можно начинать с префикса `sz`, а численные значения - с префикса `n`.

Помимо переменной `szHelloMsg`, в области заголовка документа HTML с помощью ключевого слова `function` определена функция с именем `printHello`. Эта функция вызывается из сценария, расположенного в теле документа и выводит в документ HTML значение переменной `szHelloMsg`.

Интерпретация документа HTML и встроенных в него сценариев происходит по мере загрузки документа. Поэтому если в сценарии одни функции вызывает другие или используют определенные в документе переменные, то их определения (вызываемых функций и переменных) необходимо разместить выше вызывающих. Размещение определений переменных и функций в разделе заголовка документа гарантирует, что они будут загружены до момента загрузки тела документа.

Вариация четвертая: с диалоговой панелью сообщения

Язык JavaScript имеет встроенные средства для отображения простейших диалоговых панелей, таких как панель сообщений. В примере 1.6 приведен исходный текст сценария JavaScript, в котором вызывается функция `alert`, предназначенная для отображения диалоговых панелей с сообщениями.

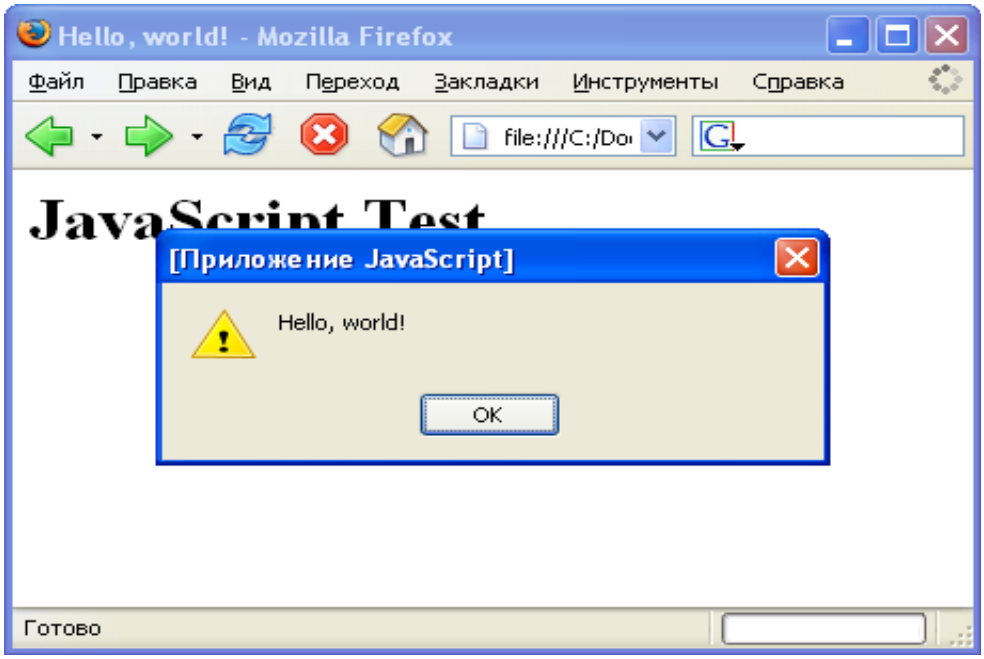
Пример 1.6.

```
<HTML>
  <HEAD>
    <TITLE>Hello, world!</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    function printHello()
    {
      alert("Hello, world!");
    }
    // -->
    </SCRIPT>
  </HEAD>
  <BODY>
    <H1>JavaScript Test</H1>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    printHello();
    // -->
```

```
    </SCRIPT>
  </BODY>
</HTML>
```

Результат работы сценария JavaScript в рассмотренном документе приведен на рисунке 1.2.

Рисунок 1.2. Диалоговая панель сообщения



Помимо представленной в этом примере диалоговой панели сценарии JavaScript могут выводить на экран и более сложные. В них пользователь может делать, например, выбор из двух альтернатив или даже вводить какую-либо информацию.

Вариация пятая: с диалоговой панелью ввода информации

В данном примере рассматривается использование диалоговой панели ввода информации. Введенная в диалоговой панели текстовая строка выводится в окне браузера.

Пример 1.7.

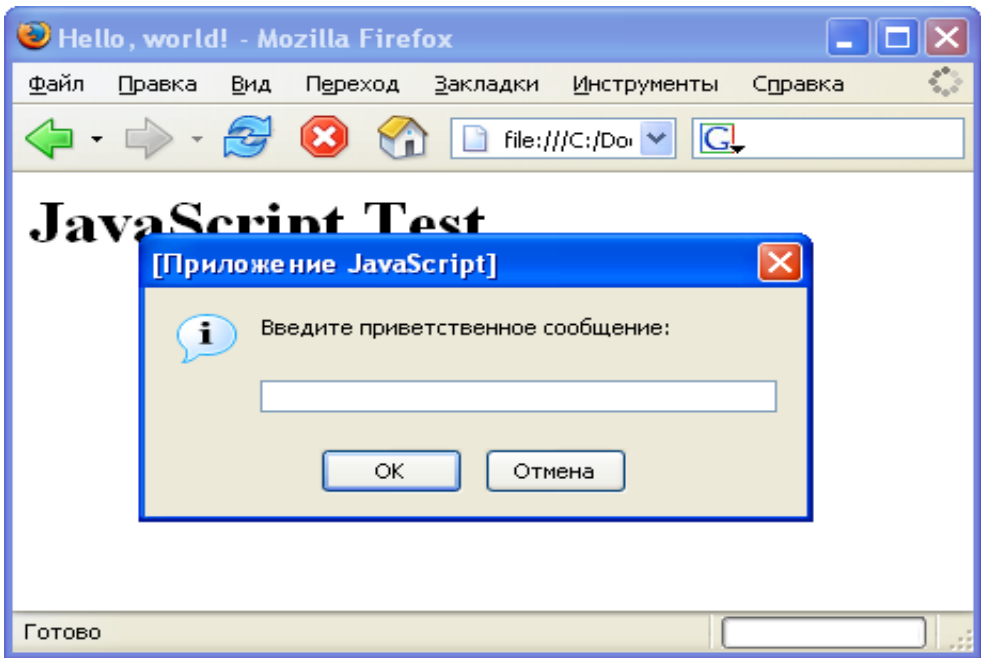
```
<HTML>
  <HEAD>
    <TITLE>Hello, world!</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    function printHello()
    {
      szHelloStr=prompt("Введите приветственное
сообщение:", "");
```

```
      document.write(szHelloStr);
    }
    // -->
    </SCRIPT>
  </HEAD>
  <BODY>
    <H1>JavaScript Test</H1>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    printHello();
    // -->
    </SCRIPT>
  </BODY>
</HTML>
```

Диалоговая панель ввода информации вызывается с помощью функции `prompt`. В качестве параметров функции передается вводное сообщение для пользователя и начальное значение запрашиваемой текстовой строки (в приведенном примере - пустое).

Пример диалоговой панели ввода информации приведен на рисунке 1.3.

Рисунок 1.3. Диалоговая панель ввода информации



Вариация шестая: обработка события

В языке JavaScript есть удобные средства обработки событий. В следующем примере, когда пользователь пытается выбрать ссылку "Select me!", разместив над ней курсор мыши, на экране появляется диалоговая

панель с сообщением "Hello, world!". Исходный текст соответствующего документа HTML с встроенным в него сценарием представлен в примере 1.8.

Пример 1.8.

```
<HTML>
  <HEAD>
    <TITLE>Hello world!</TITLE>
  </HEAD>
  <BODY>
    <H1>JavaScript Test</H1>
    <A HREF="" onMouseover="alert('Hello, world!');">Select
me!</A>
  </BODY>
</HTML>
```

Здесь для нас интересна строка тега <A>. Напомним, что этот тег применяется для организации ссылок на другие документы HTML или файлы различных объектов. В данном случае поле ссылки параметра HREF пустое, однако дополнительно в тег <A> включена следующая конструкция:

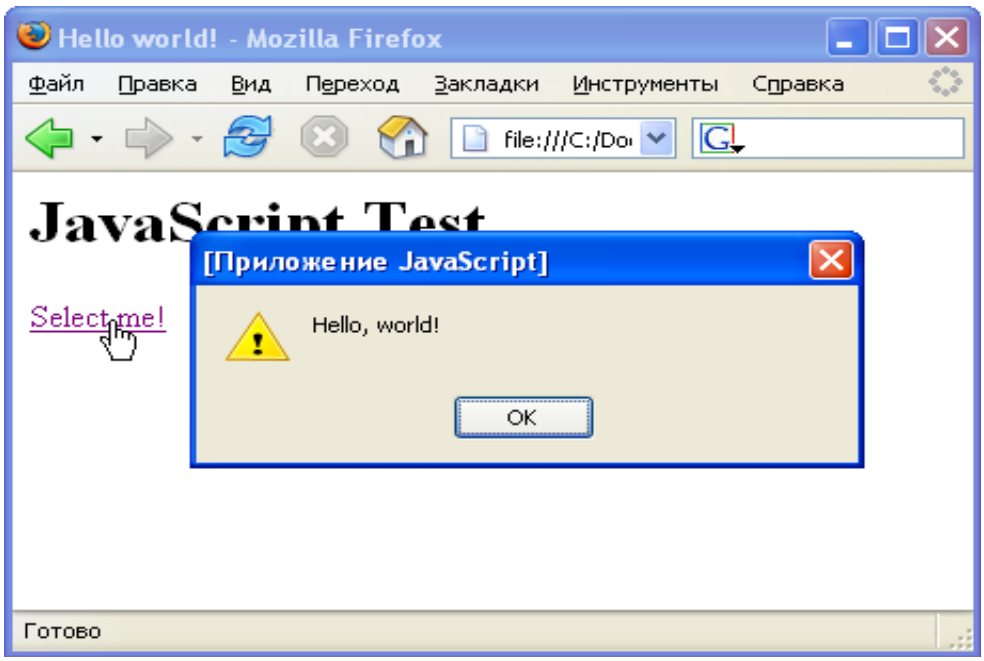
```
onMouseover="alert('Hello, world!');"
```

Она указывает, что при возникновении события onMouseover (наведение мыши на объект) должна выполняться следующая строка программы JavaScript:

```
alert('Hello, world!');
```

Результат работы сценария, встроенного подобным образом в документ HTML, представлен на рисунке 1.4.

Рисунок 1.4. Обработка события “наведение мыши”



Обратите внимание, что строка `'Hello, world!'` задана не в двойных кавычках, а в одинарных. В сценариях JavaScript допустимо использовать и те, и другие кавычки (при условии, что закрывающая кавычка должна быть такой же, как и открывающая), что позволяет избежать путаницы при использовании вложенных конструкций. Так, в нашем примере, строка `'Hello, world!'` является составной частью более общей строки `"alert('Hello, world!');"`. Как видим, внутренняя пара кавычек отличается от внешней, что позволяет правильно определить структуру фрагмента сценария JavScript.

Задания для самостоятельного выполнения

1.1. Измените пример 1.1 так, чтобы фраза “Hello, world!” выводилась в окно браузера в оформленном виде (цвет, размер, стилевое выделение и т.п.). Что для этого необходимо изменить, документ HTML, или включенный в него сценарий JavaScript?

1.2. Создайте документ с диалоговой панелью ввода информации (пример 1.7), проверьте его работу. В предлагаемую диалоговую панель введите фразу, включающую в себя теги HTML (например, *Привет*, *Внимание!*, *<img src=”http://www.fizmat.vspu.ru/pic/mf-pic.gif”>*. Как браузером обрабатывается введенный текст?

1.3. Измените пример 1.8 так, чтобы диалоговая панель возникала не при наведении курсора мыши, а *при выборе ссылки*. (Необходимо установить

обработчик события `onClick`. Чтобы при выборе гиперссылки переход на новую страницу не происходил, в теге `<A>` укажите в качестве значения параметра `href` строку "`javascript:void(0)`").

1.4. Измените пример 1.8 так, чтобы при наведении курсора мыши на ссылку, выполнялась бы процедура, выводящая фразу "Hello, world!" *в окно браузера*. Как изменяется вид страницы, отображаемой в окне браузера?

1.5. Напишите сценарий, который сначала выводит на экран диалоговое окно, а затем, после нажатия кнопки "ОК", в окне браузера пишет фразу "Hello, world!".

1.6. Напишите сценарий, который запрашивает у пользователя информацию, а затем выводит ее *в новом диалоговом окне*.

1.7. Составьте документ так, чтобы диалоговое окно для ввода информации предлагалось только *после выбора некоторой ссылки*, и введенная пользователем текстовая строка выводилась бы в виде нового диалогового окна.

Занятие 2. Основы построения языка JavaScript

Переменные в JavaScript

В сценариях JavaScript можно использовать переменные, адресуясь к ним по имени. Переменные могут быть как глобальные, так и локальные. Глобальные переменные доступны из любого места сценария. Область действия локальных переменных ограничивается функцией, внутри которой эти переменные объявлены. Так же как и в языке программирования Basic, при составлении сценариев JavaScript можно использовать переменные без их предварительного объявления. Исключение из этого правила - локальные переменные, определенные в функциях. Тем не менее, рекомендуется объявлять переменные перед их использованием, а также присваивать им начальные значения. Это упрощает отладку сценариев и уменьшает вероятность возникновения ошибок при составлении исходного текста, особенно если одновременно применяются глобальные и локальные переменные.

Объявление переменных

Все переменные в JavaScript объявляются с помощью ключевого слова `var`, как это показано ниже:

```
var szHelloMsg;
```

Тип переменной, как правило, определяется в момент первого присвоения значения (подробнее будет рассмотрено в следующих разделах).

При выборе имен переменных необходимо придерживаться следующих правил:

1. Имя переменной должно начинаться с буквы или с символов "_", "$" и может состоять только из букв, цифр, а также символов "_", "$";
2. Регистр букв **имеет значение**. Так, например, `nValue` и `nvalue` – **разные** переменные!
3. Имя переменной не должно совпадать с зарезервированными ключевыми словами JavaScript. Список зарезервированных ключевых слов JavaScript приведен ниже:

```
break    case      catch    class     const    continue
debugger default   delete   do        else     enum
export   extends   false    finally   for      function
if       import    in       new       null     return
super    switch    this     throw     true     try
typeof   var       void     while     with
```

Среди этих слов есть такие, которые еще только планируется применять в языке JavaScript при его развитии. Тем не менее, хотя их использование в качестве имен переменных в настоящее время все-же возможно, делать этого настоятельно не рекомендуется. Нужно также следить, чтобы имена переменных не совпадали с именами встроенных объектов, методов и функций.

Присвоение значения переменным

Значение переменной присваивается при помощи оператора присвоения "=". В качестве примера, ниже объявлена переменная и затем в нее записана текстовая строка (в результате чего и сама переменная становится строкового типа):

```
var szHelloMsg;
szHelloMsg = "Hello, world!";
```

Язык JavaScript допускает сокращенную запись объявления переменной и присвоения ей первого значения. Так, приведенный выше пример в сокращенном виде можно записать следующим образом:

```
var szHelloMsg = "Hello, world!";
```

В любом месте программы можно присвоить переменной `szHelloMsg` численное значение, например, так:

```
szHelloMsg = 4;
```

После выполнения такой операции *тип переменной изменится*, причем в процессе интерпретации сценария браузер не отобразит никаких предупреждающих сообщений, так как такое изменение является допустимым.

Типы данных в JavaScript

В языке JavaScript существует несколько типов данных. Это числа, текстовые строки, логические данные, объекты, данные неопределенного типа, а также специальный тип null. Объекты будут рассмотрены позже, а сейчас обратим внимание на остальные типы данных.

Числа

Язык сценариев JavaScript допускает использование чисел в различных форматах. Это целые числа, числа в формате с плавающей десятичной точкой и числа в научной нотации (с указанием десятичного порядка). Целые числа могут быть представлены по основанию 8, 10 или 16. Например:

```
25         Целое число по основанию 10
0137       Целое число по основанию 8
0xFF       Целое число по основанию 16
386.7      Число с плавающей десятичной точкой
25e5
или 25E5   Число в научной нотации, равно 2500000
```

В некоторых случаях арифметические функции могут возвращать так называемое "нечисло", которое называется в JavaScript как NaN (Not a Number). "Нечисло" - это специальное значение, которое не соответствует никакому числу. Оно возникает в тех случаях, когда результат выполнения операции над числами не может быть представлен в виде числа. С помощью функции isNaN можно проверить, является ли значение "нечислом".

Тип null

Переменная может иметь специальное значение `null`:

```
szHelloMsg = null;
```

Такое присвоение не назначает переменной никакого типа. Оно применяется в тех случаях, когда нужно объявить переменную и проинициализировать ее, не присваивая этой переменной никакого начального значения и типа.

Текстовые строки

Текстовая строка - это последовательность символов, заключенных в одинарные или двойные кавычки, например:

```
"Hello, world!"
""
"12345"
'Это текстовая строка'
```

Строка "" - пустая. Заметим, что следующие два присвоения не эквивалентны:

```
szStr=""
szStr1=null
```

В первом случае в переменной `szStr` хранится текстовая строка (хотя бы и пустая), во втором - совсем ничего.

Логические данные

Логические данные могут иметь только два значения: `true` и `false`. Эти значения никак не соотносятся с числами `1` и `0`. Они предназначены главным образом для выполнения операций сравнения, а также для использования в условных операторах.

Данные неопределенного типа

Если переменная была объявлена, но ей еще ни разу не присваивалось значение, она имеет неопределенный тип. Например, в следующей строке сценария объявлена переменная `MyVariable`, которая имеет неопределенный тип:

```
var MyVariable;
```

Если же этой переменной присвоить значение `null`, то тип переменной изменится - теперь это будет переменная, содержащая значение `null`:

```
MyVariable = null;
```

Преобразование типов данных

Когда в выражениях встречаются переменные разных типов, интерпретатор JavaScript может автоматически преобразовывать численные данные в текстовые строки. Обратное же преобразование (строк в числа) приходится выполнять с помощью специальных функций, таких как `parseInt` и `parseFloat`. Поясним это на примере (пример 2.1).

Пример 2.1. Преобразование типов

```
<HTML>
  <HEAD>
    <TITLE>Пример преобразования типов</TITLE>
  </HEAD>
  <BODY>
    <H1> Пример преобразования типов </H1>
    <TABLE>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
      var szTextBuf = "";
      szTextBuf = 4 + " - число четыре" + "<BR>";
      szBuf2 = (parseInt("2") + 2) + " - число четыре" +
"<BR>";
      document.write(szTextBuf + szBuf2);
    // -->
    </SCRIPT>
    </TABLE>
  </BODY>
</HTML>
```

Здесь мы объявили переменную `szTextBuf` и проинициализировали ее пустой строкой. Далее мы присвоили этой строке сумму числа `4` и двух текстовых строк:

```
szTextBuf = 4 + " - число четыре" + "<BR>";
```

При вычислении этого выражения значение `4` автоматически преобразовывается в текстовую строку. Дальнейшее суммирование выполняется как слияние (конкатенация) трех текстовых строк. В следующей строке преобразовывается текстовая строка `"2"` в численное значение с помощью функции `parseInt`, прибавляется к результату преобразования число `2`, а затем выполняется конкатенацию с двумя текстовыми строками:

```
szBuf2 = (parseInt("2")+2)+" - число четыре"+ "<BR>";
```

В результате переменная `szBuf2` будет содержать ту же самую строку, что и переменная `szTextBuf`.

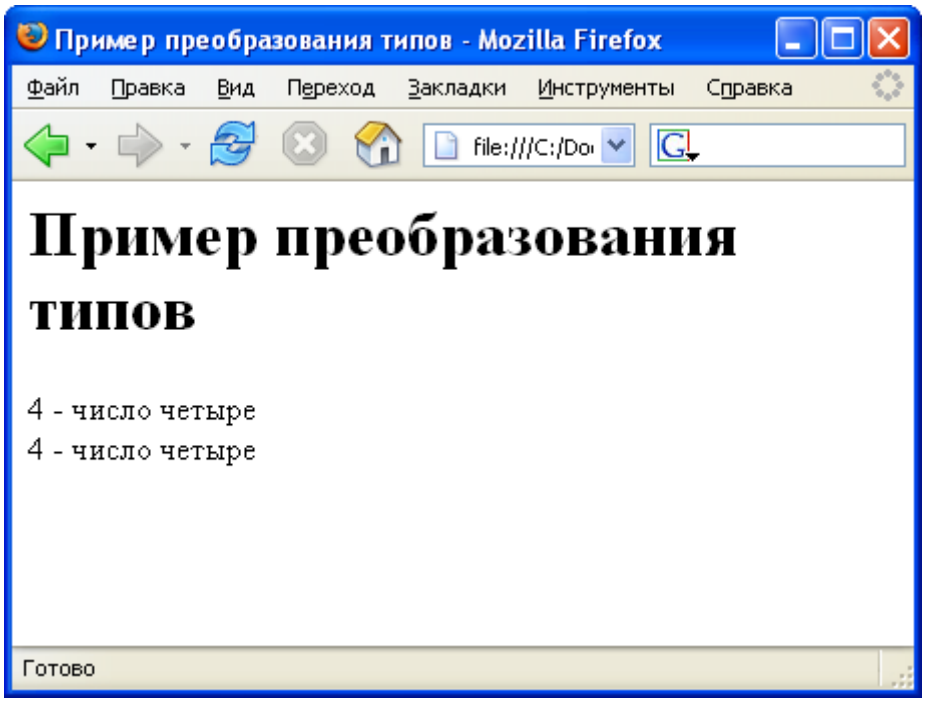
Обратите также внимание на формат вызова процедуры вывода текста в окно браузера:

```
document.write(szTextBuf + szBuf2);
```

Здесь в качестве параметра передается сумма (объединение) двух текстовых строк, то есть одна объединенная строка, которую и надо вывести в окно браузера.

Результат работы рассмотренного сценария приведен на рисунке 2.1

Рисунок 2.1. Пример преобразования типов



Операторы языка JavaScript

Унарные операторы

Унарные операторы применяются для изменения знака, выполнения операции дополнения, инкремента и декремента:

```
-    Изменение знака на противоположный
!    Дополнение. Используется для реверсирования значения
     логических переменных
++   Увеличение значения переменной. Может применяться как
     префикс переменной или как ее суффикс
--   Уменьшение значения переменной. Может применяться как
     префикс переменной или как ее суффикс
```

Ниже представлены примеры использования унарных операторов:

```
i=0;       // начальное значение переменной i равно 0
i++;       // значение i равно 1
--i;       // значение i снова равно 0

var j=3;   // значение переменной j равно 3
i = -j;    // значение переменной i равно -3

var fYes = true; // значение переменной fYes равно true
testFlag(!fYes); // функции testFlag передается значение
false
```

Бинарные операторы

Бинарные операторы соединяют два операнда. В языке сценариев JavaScript предусмотрены бинарные операторы для вычитания, сложения, умножения, деления и вычисления остатка деления:

```
-     Вычитание
+     Сложение
*     Умножение
/     Деление
%     Вычисление остатка от деления
```

Эти операторы используются таким же образом, что и в языке программирования C, например:

```
i=0;          // начальное значение переменной i равно 0
i=i+1;        // значение i равно 1

var j=9;      // значение переменной j равно 9
i=j/2;        // значение переменной i равно 4
k=j%2;        // значение переменной i равно 1
```

Операторы сдвига

Для выполнения операций сдвига в языке JavaScript предусмотрено три оператора:

```
>>    Сдвиг в правую сторону
<<    Сдвиг в левую сторону
>>>   Сдвиг в правую сторону с заполнением освобождаемых
      разрядов нулями
```

Перед использованием операторов сдвига значение переменной преобразуется в 32-разрядное целое число.

Ниже приведен пример, в котором в переменную `nValue` записывается значение, полученное в результате сдвига бит числа `4`. Сдвиг выполняется на два бита влево:

```
var nValue;
nValue = 4 << 2;
```

В результате выполнения этого фрагмента сценария переменная `nValue` принимает значение `16`.

Операторы отношения

Операторы отношения используются для сравнения значений переменных. Эти операторы возвращают логические значения `true` или `false` в зависимости от результата сравнения и применяются главным образом в условных операторах. Ниже представлен список операторов отношения языка сценариев JavaScript с указанием условия, при котором оператор возвращает значение `true`:

```
>    Левый операнд больше правого
>=   Левый операнд больше или равен правому
<    Левый операнд меньше правого
<=   Левый операнд меньше или равен правому
==   Левый операнд равен правому
!=   Левый операнд не равен правому
```

Логические операторы

Логические операторы определяют операции над логическими переменными.

```
||   Оператор ИЛИ. Возвращает значение true,
     когда один из операндов равен true
&&   Оператор И. Возвращает значение true,
     когда оба операнда равны true
```

К логическим операторам относится и рассмотренный выше унарный оператор `!` (НЕ), изменяющий значения логических переменных на противоположные.

Оператор присваивания

Оператор присваивания нами уже рассматривался и он применяется для присваивания значений переменным. Вместе с тем, в языке JavaScript, так же как и в языке программирования C, допускается комбинирование этого оператора с другими для изменения содержимого переменных. Ниже

перечислены все возможные комбинации оператора присваивания и других операторов:

```
=     Простое присваивание
+=    Увеличение численного значения или слияние строк
-=    Уменьшение численного значения
*=    Умножение
/=    Деление
%=    Вычисление остатка от деления
>>=   Сдвиг вправо
>>>=  Сдвиг вправо с заполнением освобождаемых разрядов
      нулями
<<=   Сдвиг влево
|=    ИЛИ
&=    И
^=    ИСКЛЮЧАЮЩЕЕ ИЛИ
```

Для тех, кто не знаком с языком C, комбинации оператора присваивания с другими операторами могут показаться непривычными и сложными для использования. На самом деле они упрощают сценарии, сокращая листинги исходных текстов.

Рассмотрим, например, применение оператора `+=` для увеличения содержимого числовой переменной. Вначале решим эту задачу без использования данного оператора. Ниже объявлена переменная с именем `nCounter` и присвоено ей начальное значение `1`, а затем увеличено это значение на `5`:

```
var nCounter = 1;
nCounter = nCounter + 5;
```

Теперь сделаем то же самое, но с использованием комбинаций операторов, что позволит несколько сократить запись:

```
var nCounter = 1;
nCounter += 5;
```

Для того чтобы сдвинуть содержимое переменной на три разряда вправо, можно воспользоваться оператором `>>=`, как это сделано в следующем фрагменте исходного текста:

```
nCounter >>= 3;
```

Результат при этом будет такой же, как и при выполнении следующей строки:

```
nCounter = nCounter >> 3;
```

Функции в языке JavaScript

Можно оформлять фрагменты исходного текста в виде функций, вызывая их по мере необходимости из различных мест сценария JavaScript. Обычно функции определяются в разделе заголовка документа HTML, отмеченного операторами `<HEAD>` и `</HEAD>`. Функции должны быть определена перед вызовом, и размещение всех определений функций в разделе заголовка документа HTML гарантирует доступность этих функций при обработке документа.

Общий вид определения функции представлен ниже:

```
function имя([параметр 1] [,параметр 2] [...,параметр N])
{
  . . .
  строки тела функции
  . . .
  [return значение]
}
```

Все параметры передаются функциям по значению. Поэтому функция не может изменить содержимое переменных, передаваемых ей в качестве параметров.

С помощью ключевого слова `return` функция может вернуть значение.

Первый пример использования функции нами рассматривался в «Вариации третьей» предыдущего раздела (пример 1.4). Функции являются мощным средством структурирования программ и мы будем их использовать практически во всех разрабатываемых нами сценариях.

Занятие 3. Базовые алгоритмические конструкции

Организация условных переходов

Любой язык программирования был бы бесполезен, если бы в нем не были предусмотрены те или иные средства ветвления при выполнении программы. В языке JavaScript предусмотрен условный оператор `if..else`, который позволяет выполнять разные программные строки в зависимости от условия.

Общий вид оператора `if..else` представлен ниже:

```
if(условие)
  строка 1
```

```
[else
  строка 2]
```

Часть оператора, выделенная квадратными скобками, является необязательной.

При выполнении этого оператора оценивается условие, заданное *в круглых скобках* после ключевого слова `if`. Если в результате оценки условия получилось логическое значение `true`, выполняется `строка 1`. Если же получилось значение `false`, то выполняется `строка 2` (в том случае, когда она присутствует).

Необходимо учитывать, что если в `строке 1` или `строке 2` располагается несколько операторов, то следует использовать **составной оператор** (выделить строки фигурными скобками). В приведенном ниже фрагменте сценария оценивается возраст посетителя некоторого сайта, на основании чего принимается решение о запрете или о разрешении доступа к материалам сайта, указывается следующая страница, которая должна быть представлена пользователю в соответствии с его уровнем доступа:

```
if(nYourAge < 18)
{
  bAccessDenied = true;
  szNextPage = "bye18.html";
}
else if(nYourAge > 99)
{
  bAccessDenied = true;
  szNextPage = "bye99.html";
}
else
{
  bAccessDenied = false;
  szNextPage = "welcome.html";
}
```

Здесь вначале оценивается условие `(nYourAge < 18)`. Если содержимое переменной `nYourAge` меньше `18`, переменной `bAccessDenied` присваивается значение `true`, а переменной `szNextPage` - текстовая строка `"bye18.html"`.

В противном случае содержимое `nYourAge` сравнивается с числом `99`. Если переменная `nYourAge` имеет значение, большее чем `99`, в переменную `bAccessDenied` записывается значение `true`, а переменную `szNextPage` - текстовая строка `"bye99.html"`.

И, наконец, если ни одно из двух условий не было выполнено, то есть значение переменной `nYourAge` находится в интервале от `18` до `99`, в

переменную bAccessDenied записывается значение false, а в переменную szNextPage - текстовая строка "welcome.html".

Существует также специальный тип условного оператора, который называется оператором ?:. Этот оператор в общем виде записывается так:

```
выражение ? строка 1 : строка 2
```

При вычислении оператора ?: вначале оценивается логическое выражение, расположенное в левой части. Если оно равно true, выполняется строка 1, а если false - строка 2.

Ниже приведен пример использования условного оператора ?: для присвоения значения переменной bAccessDenied в зависимости от содержимого переменной nYourAge:

```
bAccessDenied =
    (nYourAge < 18 || nYourAge > 99) ? true : false;
```

Если значение переменной nYourAge находится в интервале от 18 до 99, переменной bAccessDenied присваивается значение true, а если оно не попадает в этот интервал - false. Традиционное решение этой задачи с помощью оператора else..if заняло бы места значительно больше:

```
if(nYourAge < 18 || nYourAge > 99)
  bAccessDenied = true;
else
  bAccessDenied = false;
```

Задания для самостоятельного выполнения

3.1. Напишите сценарий, который запрашивает два числа и выводит в окне браузера наибольшее из них.

3.2. Напишите сценарий, который запрашивает номер месяца и выводит название времени года.

3.3. Используя рассмотренный фрагмент сценария определения уровня доступа составьте сценарий, запрашивающий ваш возраст и выдающий в окне браузера текст «Доступ закрыт, вам меньше 18-и», «Доступ закрыт, вам больше 99-и», «Доступ открыт».

3.4. Напишите сценарий, который отображает диалоговое окно с кнопками "Ок" и "Отмена" (функция confirm) и выводит в окно браузера сообщение о том, какая из кнопок была нажата (используйте конструкцию if(confirm("Сообщение")) ...)

Операторы цикла

В языке JavaScript есть несколько операторов, предназначенных для организации циклов.

Оператор for

Общий вид оператора `for` представлен ниже:

```
for ([инициализация;] [условие;] [итерация])
{
  . . .
  строки тела цикла
  . . .
}
```

В области инициализации обычно выполняется присваивание начальных значений переменным цикла. Здесь допустимо объявление новых переменных при помощи ключевого слова `var`. Вторая область задает условие выхода из цикла. Это условие оценивается каждый раз при прохождении цикла. Если в результате оценки получается логическое значение `true`, выполняются строки тела цикла. Область итерации применяется для изменения значений переменных цикла, например, для увеличения счетчика цикла.

Пример использования цикла `for` представлен ниже. Данный цикл 10 раз выводит приветственную фразу в окно браузера.

```
szString = "Привет!";
for (i = 0; i < 10; i++)
{
  document.write(szString + "<BR>");
}
```

Для тех, кто знаком с языками Basic и Pascal, подобная форма организации арифметического цикла может показаться неудобной. Однако, во-первых, следует иметь в виду, что здесь возможно более “тонкое” управление переменными цикла (она, кстати, может быть не одна), а во-вторых, для организации перебора элементов массива, свойств объектов существует специальная (упрощенная) форма цикла `for`, которая рассмотрена ниже.

Оператор for-in

Оператор `for-in` предназначен для просмотра всех свойств объекта и записывается в следующем виде:

```
for (переменная in объект)
```

```
{
  . . .
  строки тела цикла
  . . .
}
```

Так, например, для перебора и вывода всех элементов некоторого массива (массивы в JavaScript являются объектами и подробно будут рассмотрены позже) удобнее всего использовать конструкцию следующего вида:

```
for (n in myArray)
{
  document.write(n + "<br>");
}
```

Заметим, что при использовании оператора `for-in` не возникает даже необходимости в специальном определении размера массива, это делается автоматически.

Оператор while

Для организации итерационных циклов с предусловием используется оператор `while`:

```
while(условие)
{
  . . .
  строки тела цикла
  . . .
}
```

Если в результате оценки условия получается значение `true`, тогда итерация выполняется, если `false` - цикл прерывается.

В качестве примера рассмотрим уже знакомую задачу десятикратного вывода приветственного сообщения в окно браузера:

```
szString = "Привет!";
i = 0;
while (i < 10)
{
  document.write(szString + "<BR>");
  i++;
}
```

Оператор break

С помощью оператора `break` можно прервать выполнение цикла, созданного операторами `for` или `while`, в любом месте. Например:

```
var i = 0;
while(i < 10)
{
  . . .
  i++;
  if(i+n = 0)
    break;
  . . .
}
```

Использование оператора break позволяет восполнить недостаток специальной конструкции, позволяющей организовать **итерационный цикл с постусловием**. Подобный цикл можно организовать так:

```
while(true)
{
  . . .
  строки тела цикла
  . . .
  if(условие)
    break;
}
```

Оператор continue

Выполнение оператора `continue` внутри цикла `for` или `while` приводит к тому, что текущая итерация прерывается и начинается новая. Этот оператор не прерывает цикл. Ниже приведен пример использования оператора `continue`:

```
var i = 0;
while(i < 100)
{
  i++;
  if(i < 10)
    continue;
  . . .
}
```

Здесь фрагмент тела цикла, отмеченный многоточием, будет выполняться только после того, как значение переменной `i` станет равным `10`. Когда же это значение достигнет `100`, цикл будет завершен.

Задания для самостоятельного выполнения

3.5. Напишите сценарий, запрашивающий количество учащихся студенческой группы и в соответствии с этим количеством - их фамилии и имена. Фамилии и имена необходимо отобразить в окне браузера.

3.6. Напишите сценарий, который запрашивает фамилии и имена до тех пор, пока пользователь в окне ввода не нажмет кнопку "Отмена". Фамилии и имена необходимо отобразить в окне браузера.

3.7. Напишите сценарий, выводящий в окно браузера таблицу умножения

3.8. Создайте в своей папке несколько графических файлов с именами `pic1.gif`, `pic2.gif`, `pic3.gif` и т.д. Напишите сценарий, который с помощью цикла отобразит все эти изображения в окне браузера.

3.9. Напишите сценарий, который отобразит в окне браузера доску для игры в шахматы.

Занятие 4. Встроенные объекты JavaScript

Язык JavaScript является объектно-ориентированным. Идея объектного подхода положена в основу языка и без использования объектов сложно написать даже простейший сценарий. Подробно объекты нами будут рассмотрены в отдельном пособии, а в рамках настоящего раздела мы остановим свое внимание лишь на трех из них, относящихся к встроенным объектам JavaScript и позволяющих выполнять такие важные операции любого языка программирования, как использование массивов, обработку строк и выполнение математических операций

Массивы в JavaScript

Массивы в JavaScript создаются как объекты встроенного класса `Array`:

```
var myArray;
myArray = new Array();
```

Здесь сценарий JavaScript создает объект `myArray`, применяя для этого ключевое слово `new`, и конструктор `Array()` без параметров.

После этого можно сразу заполнять массив значениями:

```
myArray[0] = 8;
myArray[1] = 25;
myArray[3] = "Hello";
```

Обратите внимание, что тип элементов массива не имеет значения. Он может быть различным для разных элементов. Если каждый из элементов массива объявить новым массивом, то можно получить аналог двумерного массива.

Заполнить массив значениями можно также и на стадии его создания. Делается это, например, так:

```
colors = new Array ("red","white","blue")
```

В JavaScript нет нужды беспокоиться о размере массива - он устанавливается автоматически в момент присвоения значений элементам массива. Если, например, обратиться к элементу `myArray[99]`, то размер устанавливается в 100 элементов (нумерация начинается с 0), а элементы с 4-го по 98-й (в нашем случае) остаются неопределенными. Следует отметить, что такой подход позволяет только увеличивать размер массива, но не уменьшать его (в JavaScript это невозможно).

Вместе с тем, при желании, размер массива в явном виде при его создании задать можно. Число элементов массива передается в качестве параметра конструктору `Array()`. Например массив с индексами от 0 до 49 можно объявить так:

```
myArray = new Array(50);
```

Размер (число элементов) существующего массива можно узнать с помощью свойства `length`. Например:

```
k = myArray.length;
document.write("размер массива" + k);
```

Заметим, что `length` является *свойством* объекта `myArray`, поэтому вызывается через указание составного имени (`myArray.length`), а не как обычная функция с передачей некоторого параметра для обработки.

Определение числа элементов массива бывает необходимым для построения циклических конструкций, позволяющих перебирать элементы массива. Однако напомним, что в JavaScript, существует и другой способ перебора всех элементов массива - с помощью специального вида цикла `for`:

```
for(n in myArray)
{
```

```
  document.write(n + "<br>");
}
```

Для массивов определены три метода: `join`, `reverse`, `sort`.

Метод `join` объединяет элементы массива в строку символов, в качестве аргумента в этом методе задается разделитель:

```
colors = new Array("red","white","blue")
szStr = acolors.join("+")
```

В результате выполнения присваивания значения строке символов `szStr` получаем строку `"red + white + blue"`. Этот метод удобно использовать для вывода значений всех элементов массива. Как видим, это исключает использование цикла для перебора элементов.

Метод `reverse` изменяет порядок элементов массива на обратный, а метод `sort` отсортировывает их в порядке возрастания.

Задания для самостоятельного выполнения

4.1. Напишите сценарий, который циклически запрашивает фамилии и имена некоторых людей, а затем выводит их *в алфавитном порядке* в окно браузера.

4.2. Напишите сценарий, который сортирует некоторый массив *по убыванию*.

4.3. Напишите сценарий, который циклически запрашивает целые числа и заносит их в массив. Для сформированного таким образом массива напишите функции:

- Определения числа элементов;
- Определения минимального элемента;
- Определение максимального элемента;
- Определения суммы элементов;
- Определения среднего значения.

Данные функции должны вызываться по щелчку мыши на соответствующих ссылках, представленных на странице, и выводить информацию в диалоговое окно.

Работа со строками

Мы уже немного рассматривали использование строковых переменных в JavaScript не вдаваясь в подробности их «внутреннего устройства». На самом деле, в JavaScript каждая строковая переменная – это объект класса `string`. Как видим, объекты данного класса создаются всякий раз, когда в сценарии

создаются строковые переменные. Свойства и методы данных объектов (строковых переменных) позволяют проводить различные операции со строками. Рассмотрим основные из них.

Свойства

`length`

Определяет длину строки. Пример использования:

```
var szStr;
szStr = "Hello World!";
k = szStr.length;
document.write("Длина строки " + k);
```

Методы

`charAt(index)`

Возвращает символ, указанный в `index`. Символы в строке индексируются слева направо. Индексом первого символа является `0`. Если указан `index` превышающий количество символов в строке, JavaScript возвратит пустую строку. Пример использования (будут выведены символы H и W):

```
var szStr;
szStr = "Hello World!";
document.write(szStr.charAt(0));
document.write(szStr.charAt(6));
```

`indexOf(searchValue, [fromIndex])`

Возвращает позицию первого вхождения подстроки `searchValue` в вызванном объекте. Поиск начинается с `fromIndex` (если не указано, то с начала строки). Если подстрока не найдена, то возвращается `-1`. Пример использования (будет выведено 5):

```
var szStr;
szStr = "Язык сценариев JavaScript сценариев язык";
k = szStr.indexOf("сценариев");
document.write(k);
```

`lastindexOf(searchValue, [fromIndex])`

Возвращает позицию последнего вхождения подстроки `searchValue` в вызванном объекте. Поиск начинается с `fromIndex` (если не указано, то с конца строки). Если подстрока не найдена, то возвращается `-1`. Пример использования (будет выведено 26):

```
var szStr;
szStr = "Язык сценариев JavaScript сценариев язык";
k = szStr.lastindexOf("сценариев");
document.write(k);
```

`substring(indexA, indexB)`

Если `indexA` меньше чем `indexB`, то метод `substring` возвращает подстроку, начиная с символа `indexA` и заканчивая символом *перед* `indexB`. Если `indexA` больше чем `indexB`, то метод `substring` возвращает подстроку, начиная с символа `indexB` и заканчивая символом *перед* `indexA`. Если `indexA` равен `indexB`, то метод `substring` возвращает пустую строку. Пример использования (будет выведено “сценариев”):

```
var szStr;
szStr = "Язык сценариев JavaScript сценариев язык ";
document.write(szStr.substring(5,14));
```

`toLowerCase`

Возвращает значение вызванной строки, переведенной в нижний регистр. Пример использования (будет выведено “hello world!”):

```
var szStr;
szStr = "Hello World!";
document.write(szStr.toLowerCase());
```

`toUpperCase`

Возвращает значение вызванной строки, переведенной в верхний регистр. Пример использования (будет выведено “HELLO WORLD!”):

```
var szStr;
szStr = "Hello World!";
document.write(szStr.toUpperCase());
```

Дополнительные функции для работы со строками

Ниже перечислены некоторые встроенные функций, не являющиеся методами объекта `String`, но тем не менее предназначенные для работы со строками.

`eval`

Функция `eval` предназначена для преобразования текстовой строки в численное значение. Через единственный параметр она получает текстовую строку и вычисляет ее как выражение языка JavaScript. Функция возвращает результат вычисления:

```
var nValue = Eval(szStr);
```

`parseInt`

Функция `parseInt` предназначена для преобразования текстовой строки в целочисленное значение. Строка передается функции через параметр:

```
var nValue = parseInt(szStr);
```

`parseFloat`

Функция `parseFloat` пытается преобразовать текстовую строку в число с плавающей точкой. Текстовая строка передается этой функции через первый параметр, а основание счисления - через второй:

```
var nFloat = parseFloat(szStr, nRadix);
```

`escape`

С помощью функции `escape` сценарий JavaScript может закодировать текстовую строку с применением URL-кодировки. В этой кодировке специальные символы, такие как пробел или символ табуляции, преобразуются к следующему виду: `%XX`, где `XX` - шестнадцатеричный код символа. Пример использования этой функции:

```
var szURL = escape(szStr);
```

`unescape`

Функция `unescape` выполняет действие, прямо противоположное действию функции `unescape` - перекодирует строку из URL-кодировки в обычную текстовую строку:

```
var szStr = unescape(szURL);
```

Задания для самостоятельного выполнения

4.4. Напишите сценарий, который предлагает ввести некоторое предложение и по запросу пользователя (по щелчку на соответствующей ссылке) в диалоговом окне выводит следующие сведения:

- Длину предложения
- Количество слов в предложении
- Самое длинное слово
- Самое короткое слово
- Количество указанных пользователем букв
- Слово, в котором встречается указанное пользователем сочетание букв

4.5. Напишите сценарий, который дважды запрашивает у пользователя его имя и проверяет корректность (совпадение) введенной информации. Проверку совпадения провести с учетом того, что имя во второй раз может быть введено в другом регистре.

4.6. Напишите сценарий, который запрашивает у пользователя некоторое арифметическое выражение и выводит результат его вычисления.

4.7. Напишите сценарий, который запрашивает текстовую строку и выводит ее в окно браузера, выделяя при этом (цветом или стилем написания) все слова “JavaScript”.

Математические вычисления в JavaScript

Хотя сценарии JavaScript редко применяют для математических вычислений, в JavaScript все же есть все для этого необходимое. Математические константы и функции реализованы как свойства и методы объекта `Math`. Объект `Math` является встроенным и его можно использовать без предварительного объявления.

Свойства объекта Math

Ниже перечислены свойства объекта `Math`. Все эти свойства являются математическими константами, поэтому сценарии JavaScript не могут изменять их значение.

E

Это свойство представляет собой константу e, приблизительное значение которой равно 2.718281828459045. Пример использования свойства E:

```
var nE;
nE = Math.E;
```

PI

Свойство PI - это число π. Оно также является константой с приблизительным значением, равным 3.141592653589793. Пример использования свойства PI (вычисления длины окружности по ее радиусу):

```
var nL;
var nR;
nL = 2 * Math.PI * nR;
```

LN2

Свойство LN2 - константа со значением натурального логарифма числа 2, то есть ln2. Пример использования:

```
var nValue;
nValue = Math.LN2;
```

LN10

Свойство LN10 - константа со значением натурального логарифма числа 10, то есть ln10. Пример использования:

```
var nValue;
nValue = Math.LN10;
```

LOG2E

Это свойство является константой со значением, равным логарифму числа 2 по основанию e, то есть loge2. Пример использования:

```
var nValue;
nValue = Math.LOG2E;
```

LOG10E

Свойство LOG10E - это логарифм числа e по основанию 10, то есть log10e. Пример использования:

```
var nValue;
nValue = Math.LOG10E;
```

QRT2

Свойство SQRT2 - это значение квадратного корня из 2. Пример использования:

```
var nValue;
nValue = Math.SQRT2;
```

SQRT1_2

Свойство SQRT1_2 - это значение квадратного корня из 0,5. Пример использования:

```
var nValue;
nValue = Math.SQRT1_2;
```

Методы

abs

Вычисление абсолютного значения. Пример использования (в переменную nValueAbs записывается абсолютное значение переменной nValue):

```
var nValueAbs;
nValueAbs = Math.abs(nValue);
```

acos

Вычисление арккосинуса. Пример использования:

```
var nValue;
nValue = Math.acos(nAngle);
```

asin

Вычисление арксинуса. Пример использования:

```
var nValue;
nValue = Math.asin(nAngle);
```

atan

Вычисление арктангенса. Пример использования:

```
var nValue;
nValue = Math.atan(nAngle);
```

ceil

Вычисление наименьшего целого значения, большего или равного аргументу функции. Пример использования:

```
var nValue;
nValue = Math.ceil(nArg);
```

`cos`
Вычисление косинуса. Пример использования:

```
var nValue;
nValue = Math.cos(nAngle);
```

`exp`
Экспоненциальная функция, значение которой равно числу e, возведенному в степень аргумента функции. Пример использования:

```
var nValueExp;
nValueExp = Math.exp(nValue);
```

`floor`
Вычисление наибольшего целого значения, меньшего или равного аргументу функции. Пример использования:

```
var nValue;
nValue = Math.floor(nArg);
```

`log`
Вычисление натурального логарифма аргумента функции. Пример использования:

```
var nValue;
nValue = Math.log(nArg);
```

`max`
Определение наибольшего из двух значений. Пример использования:

```
var nValue1;
var nValue2;
var nValueMax;
nValueMax = Math.max(nValue1, nValue1);
```

`min`
Определение наименьшего из двух значений. Пример использования:

```
var nValue1;
var nValue2;
var nValueMin;
nValueMin = Math.min(nValue1, nValue1);
```

`pow`
Возведение числа в заданную степень. Пример использования (число `2` возводится в степень `3`, а результат, равный `8`, записывается в переменную `nValue`):

```
var nValue;
nValue = Math.pow(2, 3);
```

`random`

Метод `random` возвращает случайное число в интервале от `0` до `1`. Пример использования:

```
var nRandomValue;
nRandomValue = Math.random();
```

`round`

Метод `round` предназначен для выполнения округления значения аргумента до ближайшего целого. Если десятичная часть числа равна 0,5 или больше этого значения, то выполняется округление в большую сторону, если меньше - в меньшую. Пример использования (после выполнения округления значение nValue будет равно 2):

```
var nValue;
nValue = Math.round(1.8);
```

`sin`

Вычисление синуса. Пример использования:

```
var nValue;
nValue = Math.sin(nAngle);
```

`sqrt`

Вычисление квадратного корня от аргумента. Пример использования:

```
var nValueSqrt;
nValueSqrt = Math.sqrt(nArg);
```

`tan`

Вычисление тангенса. Пример использования:

```
var nValue;
nValue = Math.tan(nAngle);
```

Задания для самостоятельного выполнения

4.8. Измените задачи 4.2 и 4.3 так, чтобы необходимый массив создавался случайным образом.

4.9. Напишите сценарий, выводящий в окно браузера таблицу случайных целых чисел.

4.10. Напишите сценарий, формирующий таблицу квадратов (кубов, логарифмов, корней и т.п.) для некоторого набора чисел.

4.11. Используя графические изображения, созданные для задачи 3.8, напишите сценарий, который при загрузке страницы будет случайным образом выбирать одну из картинок и отображать ее в окне браузера.

Литература

1. Фролов А.В., Фролов Г.В. Сценарии JavaScript в активных страницах Web: Библиотека системного программиста, том 34. – Москва: Издательство Диалог-МИФИ, 1998, 284 с.

2. Кингсли-Хью Э., Кингсли-Хью К. JavaScript 1.5: учебный курс. – СПб.: Питер, 2002. – 272 с.

3. Вайк Аллен и др. JavaScript в примерах: Пер. с англ./Ален Вайк и др. – К.: Издательство «ДиаСофт», 2000. – 304 с.

4. Дмитриева М.В. JavaScript: простые сценарии: Заочная школа современного программирования. Занятие 1: Учебное пособие. – СПб.: Издательство ЦПО «Информатизация образования», 2003. – 27с.

5. Стефан Кох. Введение в JavaScript. – http://www.citforum.ru/internet/koch/tutorial.htm (1 дек. 2004)

6. Павел Храмцов. Практическое введение в программирование на JavaScript. - http://www.citforum.ru/internet/js_tut/index.shtml (1 дек. 2004)

7. Фролов А.В., Фролов Г.В. Сценарии JavaScript в активных страницах Web: Библиотека системного программиста, том 34. – http://proglib.ru/detail_book.asp?id=140 (1 дек. 2004)

8. Сергеев А.Н. Система уроков по JavaScript. – http://www.fizmat.vspu.ru/books/js/ (1 дек. 2004)