**Лабораторная работа №1**

**Краткие теоретические сведения.**

JavaScript является языком сценариев (скриптов), который применяют в основном для создания на Web-страницах интерактивных элементов. Его можно использовать для построения меню, проверки правильности заполнения форм, смены изображений или для чего-то еще, что можно сделать на Web-странице. Если взглянуть на GoogleMaps или службу GMail компании Google, то можно понять, на что способен сегодня язык JavaScript.

Так как JavaScript является в настоящее время единственным языком сценариев, который поддерживают все основные браузеры Web (InternetExplorer, Firefox, Netscape, Safari, Opera, Camino и т.д.), то он используется очень широко.

Код JavaScript обычно выполняется Web-браузером клиента, и в этом случае он называется сценарием на стороне клиента. Но код JavaScriptможно выполнять также на Web-сервере для формирования документов HTML, воплощая тем самым сценарий на стороне сервера. Хотя использованиеJavaScript обычно ограничивается сценариями на стороне клиента, он является также очень мощным серверным языком.

При создании кода JavaScript требуется фактически только текстовый редактор и Web-браузер. Знание HTML и CSS будет играть определенно положительную роль, и если вы захотите использовать навыки JavaScript на Web-сайте, то понадобится также Web-сайт. Если у вас уже есть Web-сайт, то отлично! Если нет, то существует множество бесплатных серверов, которые можно использовать для размещения своих страниц.

Что касается текстового редактора, то в Windows имеется редактор NotePad. Хотя этого будет достаточно для редактирования JavaScript, HTMLи CSS, более мощный редактор, такой, например, как EditPlus или другой, может оказаться более удобным.

Наиболее простой пример Javascript можно представить следующим текстом:

<SCRIPT LANGUAGE="javascript">

document.write

("<FONT COLOR='RED'>Этокрасныйтекст</FONT>")

</SCRIPT>

В результате выполнения этого сценария на странице в браузере должен появиться текст: Это красный текст.

JavaScript не может существовать сам по себе, он должен выполняться внутри Web-страницы, а Web-страница должна просматриваться в браузере, который понимает язык JavaScript (скажем, NetscapeCommunicator или InternetExplorer).

Добавить сценарий на страницу одним из двух способов: поместить теги Script на Web-странице и расположить код JavaScript внутри этих тегов, или поместить весь код JavaScript в отдельный файл и связаться с ним с помощью тега Script.

Любой из этих методов вполне допустим, но они имеют разное назначение. Если имеется небольшой код, который будет использоваться только на одной странице, то размещение его между тегами Script будет хорошим решением. Если, однако, имеется большой фрагмент кода, который будет использоваться на нескольких страницах, то, наверно, лучше поместить этот код JavaScript в отдельный файл и соединиться с ним. Это делается для того, чтобы не нужно было загружать этот код всякий раз при посещении различных страниц. Код загружается один раз, и браузер сохраняет его для последующего использования. Это похоже на то, как используются каскадные таблицы стилей (CSS).

Ниже приведены примеры двух способов подключения кода JavaScript:

<script type="text/javascript"></script>

<script type="text/javascript" src="scripts/JavaScriptFile.js"></script>

В первом примере, код JavaScript помещается между символами > и <, прямо перед "</script> ". Если вы совершенно не знаете, как работает Web-страница, то вот пример того, как устроена страница HTML:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>

<HEAD>

<TITLE> Здесь располагается заголовок страницы </TITLE>

<META NAME="Generator" CONTENT="EditPlus">

<META NAME="Author" CONTENT="Имяавтора">

<SCRIPT TYPE="text/javascript">

</SCRIPT>

</HEAD>

<BODY>

Здесь располагается основная содержательная часть Web-страницы (тело).

</BODY>

</HTML>

Обратите внимание, что текст внутри скобок находится в кавычках. В HTML эти кавычки не требуются. Здесь они необходимы. Никогда нельзя про них забывать.

Почти любой язык программирования в мире имеет дело с объектами, называемыми "переменными", и JavaScript не является исключением. Переменная является просто элементом данных с присоединенным к нему именем. Она может содержать число, слово или предложение (называемые строками - String) или объект (Object). Если коду необходимо сообщить, что имеется 5 яблок, то можно создать переменную с именем "apples" и задать ей значение 5. В JavaScript для определения переменной используется ключевое слово "var". Отметим, что JavaScript различает регистр символов, поэтому "var" означает не то же самое, что "VAR" или "Var".

varapples = 5;

Требуется помнить о том, что JavaScript является "слабо типизированным" языком. Это означает, что при определении переменных не требуется указывать, какого они типа: будут ли они числами, строками, объектами, и т.д. Во многих других языках необходимо делать это различие.

Одним из наиболее распространенных методов вывода простого сообщения пользователю является отправка уведомления (alert):

var apples = 5;

alert('Имеется ' + apples + ' яблок!');

"prompt" является другой встроенной функцией, аналогичной "alert". Однако вместо простого вывода информации она также получает ввод от пользователя.

varapples = 5;

alert('Имеется ' + apples + ' яблок!'); // Вывод на экран исходного числа яблок

vareat    = prompt('Сколько яблок вы хотите съесть?', '1'); // Запрос в отдельном окне числа съедаемых яблок

apples    -= parseInt(eat); // уменьшение общего числа яблок на указанное выше число

alert('А теперь имеется только' + apples + ' яблок!'); // вывод в отдельном окне оставшегося числа яблок

Обращение к функции "parseInt", которая получает строку и возвращает число. Так как для выполнения математических операций требуются числа, то это гарантирует, что мы имеем число. Если пользователь введет в поле "2", то parseInt превратит это в число 2.

Затем, оператор "-=", который означает вычитание из левой части оператора значения правой части. Поэтому значение переменной "eat" вычитается из переменной "apples".

Существуют другие операторы, подобные -=, которые делают похожие вещи. Всего имеется 8 обычных арифметических операторов:

+

-

/

\*

+=

-=

/=

\*=

Как и в любом языке программирования в Javascript имеются операторы цикла и ветвления.

Основной синтаксис оператора if / else следующий:

if(условие)

{

  // код, который выполняется, когда справедливо условие if

} else

{

  // код, который выполняется, когда условие if ложно

}

var apples = 5;

alert('Имеется ' + apples + ' яблок!');

vareat    = prompt('Сколько яблок вы хотите съесть?', '1');

var eaten  = parseInt(eat);

if(eaten > 5)

{

  alert('Простите, но имеется только 5 яблок.  Вы не можете съесть ' + eaten + ' яблок!');

}

else

{

  apples    -= eaten;

  alert('А теперь имеется только ' + apples + ' яблок!');

}

Операторы if и else достаточно легко понять. Приведенный выше код дает возможность сказать: "Если пользователь выбрал для еды более 5 яблок, то сообщите ему, что такого количества яблок нет. Иначе позвольте ему съесть столько яблок, сколько он попросит".

**Задание**

1.                 Создать html документ, в котором с помощью JavaScript генерируется следующий текст:

Лабораторная работа №1

Студента группы (номер группы)

Фамилия Имя Отчество (в родительном падеже)

2.                 Добавить в созданный документ сценарий, который позволял бы

a.    вычислять возраст по вводимому в отдельном окне году рождения;

b.    вычислять сумму двух вводимых в отдельных окнах чисел;

c.    вычислять стоимость весовой продукции, при вводимых в отдельных окнах цене и весе;

d.    вычислять размер стипендии в зависимости от водимого в отдельном окне среднего балла.

3.                 Оформить отчет о проделанной работе.

**Лабораторная работа №2**

**Краткие теоретические сведения.**

Иерархия классов

Объектно-ориентированный язык программирования предполагает наличие иерархии классов объектов. В JavaScript такая иерархия начинается с класса объектов   Window, т.е. каждый объект приписан к тому или иному окну. Для обращения к любому объекту или его свойству указывают полное или частичное имя этого объекта или свойства   объекта, начиная с имени объекта старшего в иерархии, в который входит данный объект:

13.2.

Вообще говоря, JavaScript не является классическим объектным языком (его еще называют облегченным объектным языком). В нем нет наследования и полиморфизма. Программист может определить собственный класс объектов черезоператорfunction, но чаще пользуется стандартными объектами, их конструкторами и вообще не применяет деструкторы классов. Это объясняется тем, что область действия JavaScript-программы обычно не распространяется за пределы текущего окна.

Иногда у разных *объектов*   *JavaScript* бывают определены *свойства* с одинаковыми именами. В этом случае нужно четко указывать, *свойство*какого *объекта* программист хочет использовать. Например, Window и Document имеют *свойство* location. Только для Window это *объект* класса Location, а для Document - строковый литерал, который принимает в качестве значения URL загруженного документа.

Управление окнами

Что можно сделать с окном? Открыть (создать), закрыть (удалить), положить его поверх всех других открытых окон (передать фокус). Кроме того, можно управлять свойствами окна и свойствами подчиненных ему объектов. Описанию основных свойств посвящен раздел "Программируем свойства окна браузера", поэтому сосредоточимся на простых и наиболее популярных методах управления окнами:

* alert();
* confirm();
* prompt();
* *open()*;
* *close()*;
* focus();
* *setTimeout()*;
* clearTimeout().

Здесь не указаны только два метода: scroll() и blur().

Первый позволяет прокрутить *окно* на определенную позицию. Но его очень сложно использовать, не зная координат окна. Последнее является обычным делом, если только не используется технология программирования слоев или CSS (CascadingStyleSheets).

Второй метод уводит фокус с окна. При этом совершенно непонятно, куда этот фокус будет передан. Лучше целенаправленно передать фокус, чем просто его потерять.

window.alert()

Метод alert() позволяет выдать *окно* предупреждения:

<A HREF="javascript:window.alert('Внимание')">

Повторите запрос!</A>

Все очень просто, но нужно иметь в виду, что сообщения выводятся системным шрифтом, следовательно, для получения предупреждений на русском языке нужна локализованная версия ОС.

window.confirm()

Метод confirm() позволяет задать пользователю вопрос, на который тот может ответить либо положительно, либо отрицательно:

<SCRIPT>

functionfunc()

{

if(window.confirm('Знаю все')==true)

{ document.getElementById("tInput").value="Учите дальше!";}

else {document.getElementById("tInput").value="Честно!";};

}

</SCRIPT>

<FORM id=form1>

<INPUT TYPE=button VALUE="Вы знаете JavaScript?"

onClick="javascript:func()"><BR>

<input tipe="text" id="tInput"  style="width:250; border:0"></input>

Все ограничения для сообщений на русском языке, которые были описаны для метода alert(), справедливы и для метода confirm().

window.prompt()

Метод prompt() позволяет принять от пользователя короткую строку текста, которая набирается в поле ввода информационного окна:

<SCRIPT>

functionfunc()

{

document.getElementById("tInput").value = window.prompt('Введитесообщение','Сюда...');

}

</SCRIPT>

<FORM id=form1>

<FORM>

<INPUT TYPE=button VALUE="Открытьокноввода"

onClick="javascript:func()">

<br>

<input tipe="text" id="tInput"  style="width:250; border:0"></input>

</FORM>

Введенную пользователем строчку можно присвоить любой переменной и потом разбирать ее в JavaScript-программе.

window.open()

У этого метода окна атрибутов больше, чем у некоторых объектов. Метод ***open()*** предназначен для создания новых окон. В общем случае его синтаксис выглядит следующим образом:

open("URL","window\_name","param,param,...", replace);

где: URL — страница, которая будет загружена в новое *окно*, window\_name — имя окна, которое можно использовать в атрибуте TARGET в контейнерах A и FORM.

|  |  |
| --- | --- |
| **Параметры** | **Назначение** |
| replace | Позволяет при открытии окна управлять записью в массив History |
| param | Список параметров |
| width | Ширина окна в пикселах |
| height | Высота окна в пикселах |
| toolbar | Создает *окно* с системными кнопками браузера |
| location | Создает *окно* с полем location |
| directories | Создает *окно* с меню предпочтений пользователя |
| status | Создает *окно* с полем статуса status |
| menubar | Создает *окно* с меню |
| scrollbars | Создает *окно* с полосами прокрутки |
| resizable | Создает *окно*, размер которого можно будет изменять |

window.close()

Метод close() — это оборотная сторона медали метода *open()*. Он позволяет закрыть *окно*. Чаще всего возникает вопрос, какое из окон, собственно, следует закрыть. Если необходимо закрыть текущее, то:

window.close();

self.close();

Если необходимо закрыть родительское *окно*, т.е. окно, из которого было открыто текущее, то:

window.opener.close();

Если необходимо закрыть произвольное *окно*, то тогда сначала нужно получить его идентификатор:

id=window.open();

...

id.close();

Как видно из последнего примера, закрывают *окно* не по имени (значение атрибута TARGET тут ни при чем), а используют указатель на объект.

Пример работы:

<SCRIPT>

function clickButton1()

{

newwindow=window.open('about:blank','test1', "width=200, height=200,resizable=no,scrollbars=no,menubar=no");

newwindow.document.write('<INPUT TYPE=button VALUE="ok" onClick="window.close()"></input>');

}

function clickButton2()

{

newwindow1=window.open('about:blank','test2',"height=400 ,width=400 ,location=yes,menubar=yes,resizable=yes,scrollbars=yes,status=yes,toolbar=yes");

newwindow1.document.write('Новая страница, сюда можно передавать параметры.');

}

</SCRIPT>

<FORM id=form1>

<INPUT TYPE=button VALUE="Открытьокноввода"

onClick="javascript:clickButton1()"></input>

<INPUT TYPE=button VALUE="Открытьокноввода"

onClick="javascript:clickButton2()"></input>

</FORM>

**Задание**

1.                 Создать html документ, в котором с помощью JavaScript:

a.    Сгенерировать следующий текст:

Лабораторная работа №2

Студента группы (номер группы)

Фамилия Имя Отчество (в родительном падеже)

b.    На страницу добавить три кнопки;

c.    При нажатии на первую кнопку, на страницу должен добавляться текст, например, отображаться текущее число, месяц, год;

d.    При нажатии на вторую кнопку должно открываться окно, содержащее текст и кнопку. Например, лицензионное соглашение с кнопкой «ОК»;

e.    При нажатии на третью кнопку должно открываться полнофункциональное окно, с несколькими полями ввода.

2.                 Оформить отчет о проделанной работе.

**Приложение**

Некоторые теги html

Label

Тег <LABEL> устанавливает связь между определенной меткой, в качестве которой обычно выступает текст, и элементом формы INPUT. Такая связь необходима, чтобы изменять значения элементов формы при нажатии курсором мыши на текст. Кроме того, с помощью <LABEL> можно устанавливать горячие клавиши на клавиатуре и переходить на активный элемент подобно ссылкам.

Синтаксис

<inputid="идентификатор"><labelfor="идентификатор">Текст</label>

<label><input type="...">Текст</label>

Существует два способа связывания объекта и метки. Первый заключается в использовании идентификатора id внутри тега INPUT и указании его имени в качестве параметра for тега <LABEL>. При втором способе тег <INPUT> помещается внутрь контейнера <LABEL>.

Input

Тег <INPUT> является одним из разносторонних элементов формы и позволяет создавать разные элементы интерфейса и обеспечить взаимодействие с пользователем. Главным образом <INPUT>предназначен для создания текстовых полей, различных кнопок, переключателей и флажков. Хотя элемент <INPUT> не требуется помещать внутрь контейнера <FORM>, определяющего форму, но если введенные пользователем данные должны быть отправлены на сервер, где их обрабатывает CGI-программа, то указывать FORM обязательно. То же самое обстоит и в случае обработки данных с помощью клиентских приложений, например, скриптов на языке JavaScript.

Основной параметр тега <INPUT>, определяющий вид элемента — type. Он позволяет задавать следующие элементы формы: текстовое поле (text), поле с паролем (password), переключатель (checkbox), флажок (radiobutton), скрытое поле (hidden), кнопка (button), кнопка для отправки формы (submit), кнопка для очистки формы (reset), поле для отправки файла (file) и кнопка с изображением (image). Для каждого элемента существует свой список параметров, которые определяют его вид и характеристики.

Синтаксис

<form>

  <input ...>

</form>

Параметры

align

Определяет выравнивание изображения.

alt

Альтернативный текст для кнопки с изображением.

border

Толщина рамки вокруг изображения.

checked

Предварительно активированный переключатель или флажок.

disabled

Блокирует доступ и изменение элемента.

maxlength

Максимальное количество символов разрешенных в тексте.

name

Имя поля, предназначено для того, чтобы обработчик формы мог его идентифицировать.

readonly

Устанавливает, что поле не может изменяться пользователем.

size

Ширина текстового поля.

src

Значение элемента.

Параметр TYPE

Описание

Сообщает браузеру, к какому типу относится элемент формы.

Синтаксис

<inputtype="button | checkbox | file | hidden | image | password | radio | reset | submit | text">

Обязательный параметр.

Аргументы

|  |  |  |
| --- | --- | --- |
| **Тип** | **Описание** | **Вид** |
| button | Кнопка. |  |
| checkbox | Флажки. Позволяют выбрать более одного варианта из предложенных. | Пиво Чай  Кофе |
| file | Поле для ввода имени файла, который пересылается на сервер. |  |
| hidden | Скрытое поле. Оно никак не отображается на веб-странице. |  |
| image | Поле с изображением. При нажатии на рисунок данные формы отправляются на сервер. |  |
| password | Обычное текстовое поле, но отличается от него тем, что все символы показываются звездочками. Предназначено для того, чтобы никто не подглядел вводимый пароль. |  |
| radio | Переключатели. Используются, когда следует выбрать один вариант из нескольких предложенных. | Пиво Чай Кофе |
| reset | Кнопка длявозвращение данных формы в первоначальное значение. |  |
| submit | Кнопка для отправки данных формы на сервер. |  |
| text | Текстовое поле. Предназначено для ввода символов с помощью клавиатуры. |  |

**Лабораторная работа №3**

**Краткие теоретические сведения.**

Наиболее зрелищные эффекты при программировании на JavaScript достигаются при работе с графикой. При этом в арсенале программиста не так уж много инструментов: встроенные в документ картинки, возможность генерации объекта Image, комбинирование картинок с гипертекстовыми ссылками и таблицами. Тем не менее обилие различных эффектов, которые достигаются этими нехитрыми средствами, впечатляет.

Программирование графики в JavaScript опирается на объект Image, который характеризуется следующими свойствами, методами и событиями:

|  |  |  |
| --- | --- | --- |
| **Свойства** | **Методы** | **События** |
| * border * complete * height * hspace * name * src * vspace * width * lowsrc | нет | * onAbort * onError * onLoad |

Все объекты класса Image можно разделить на встроенные и порожденные программистом. Встроенные объекты — это картинки контейнеров IMG. Если эти картинки поименовать, к ним можно обращаться по имени:

<A HREF="javascript:void(0);"

onClick="window.alert('Image name:'+

        document.images[0].name)">

<IMG NAME=intuit SRC=images\image.gif BORDER=0>

</A>

Картинка активна. Если на нее нажать, получим имя контейнера IMG. Обращение document.images[0].name позволяет распечатать это имя в окне предупреждения. При этом само имя указано как name=intuit в контейнере IMG.

К встроенному графическому объекту можно обратиться и по индексу:

document.images[0];

В данном случае images[0] — это первая картинка документа.

src и lowsrc

Свойства src и lowsrc определяют URL изображения, которое монтируется внутрь документа. При этом lowsrc определяет временное изображение, обычно маленькое, которое отображается, пока загружается основное изображение, чей URL указывается в атрибуте SRC контейнера IMG. Свойство src принимает значение атрибута SRC контейнера IMG. Программист может изменять значения и src, и lowsrc. Рассмотрим пример сsrc:

document.i2.src="image.gif";

Как видно из этого примера, существует возможность модифицировать вмонтированную картинку за счет изменения значения свойства srcвстроенного объекта Image. Если вы в первый раз просматриваете данную страницу (т.е. картинки не закешированы браузером), то постепенное изменение картинки будет заметно. Как ускорить это изменение, мы рассмотрим в следующем разделе.

Изменить картинку можно, только присвоив свойству src встроенного объекта Image новое значение.

Естественным продолжением идеи замещения значения атрибута SRC в контейнере IMG является мультипликация, т.е. последовательное изменение значения этого атрибута во времени. Для реализации мультипликации используют метод объекта Window — setTimeout().

Собственно, существует два способа запуска мультипликации:

  onLoad();

  onClick(), onChange()

Наиболее популярный — setTimeout() при onLoad().

Событие onLoad()

Событие onLoad() наступает в момент окончания загрузки документа браузером. Обработчик события указывается в контейнере BODY:

...

<BODY onLoad="JavaScript\_code">

...

Цикл с конечным числом замен для постепенной загрузки объемной графики может быть реализован следующим образом:

<script  language="JavaScript">

        var i=1;

        function movie()

        {

        eval("document.images[0].src='images//im"+

              i+".gif';");

        i++;

        if(i<4)

        {

               setTimeout("movie();",500);}

</script>

<BODY onLoad="movie();">

В обоих примерах следует обратить внимание на использование метода setTimeout(). На первый взгляд, это просто рекурсия. Но в действительности все несколько сложнее. JavaScript разрабатывался для многопоточных операционных систем, поэтому правильнее будет представлять себе исполнение скриптов следующим образом:

1.    Скрипт получает управление при событии onLoad().

2.    Заменяет картинку.

3.    Порождает новый скрипт и откладывает его исполнение на 500 миллисекунд.

4.    Текущий скрипт уничтожается JavaScript-интерпретатором.

Перманентная мультипликация может быть достигнута и другими средствами, например многокадровыми *графическими* файлами. Однако движение на странице — не всегда благо. Часто возникает желание реализовать запуск и остановить движения по требованию пользователя. Удовлетворим это желание, используя предыдущие примеры (запустить или остановить мультипликацию):

var flag1=0;

var i=1;

function movie()

{

if(flag1==0)

        {

               document.images[0].src='images//im'+ i+'.gif';

               i=i+1;

               if(i>3) i=1;

        }

setTimeout("movie();",1000);

}

function flag()

{

        if(flag1==0) flag1=1;

        else flag1=0;

}

...

<BODY onLoad="javascript:movie()">

<a id="test">

        <imgsrc="images/image.gif" >

</a>

<FORM>

          <input type="button" id="OK" value="OK" onClick="javascript:flag()">

</FORM>

Одним из наиболее популярных приемов дизайна страниц Web-узла является техника нарезки картинок на составные части. Можно выделить следующие способы применения этой техники для организации навигационных компонентов страницы:

  горизонтальные и вертикальные меню;

  вложенные меню;

  навигационные графические блоки.

Главной проблемой при использовании нарезанной графики является защита ее от контекстного форматирования страницы HTML-парсером. Дело в том, что он автоматически переносит элементы разметки на новую строку, если они не помещаются в одной.

Элементы переносятся на новую строку, так как ширина раздела меньше общей ширины всех картинок. Проблема решается, если применить защиту от парсера — <PRE>:

<PRE>

<IMGSRC="images\im1.gif"><IMG

SRC="images\im2.gif"><IMG

SRC="images\im3.gif">

</PRE>

Использование такого меню требует определения на нем гипертекстовых ссылок, что приводит к появлению рамки эффекту.

<PRE>

<A HREF="javascript:void(0);"><IMG

SRC="images/im1.gif"></A><A

HREF="javascript:void(0);"><IMG

SRC="images/im2.gif"></A><A

HREF="javascript:void(0);"><IMG

SRC="images/im3.gif"></A>

</PRE>

Появление рамки таблицы ссылок. Избавиться от этого визуального эффекта можно  установкой значения границы таблицы равной 0.

<PRE>

<A HREF="javascript:void(0);"><IMG

SRC="images/im1.gif" BORDER="0"></A><A

HREF="javascript:void(0);" BORDER="0"><IMG

SRC="images/im2.gif" BORDER="0"></A><A

HREF="javascript:void(0);"><IMG

SRC="images/im3.gif" BORDER="0"></A>

</PRE>

Удобнее всего организовывать разметку страницы и оформление меню в таблице.

<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0 ALIGN=center>

<TD><A HREF="javascript:void(0);"><IMG SRC="images/im1.gif" BORDER=0></A></TD>

<TD><A HREF="javascript:void(0);"><IMG SRC="images/im2.gif"BORDER=0></A></TD>

<TD><A HREF="javascript:void(0);"><IMG SRC="images/im3.gif" BORDER=0></A></TD>

  В данном случае все картинки удается сшить без пропусков и тем самым достичь непрерывности навигационного дерева. Пропуски устраняются путем применения атрибутов BORDER, CELLSPACING и CELLPADDING. Первый устраняет границы между ячейками, второй устанавливает расстояние между ячейками равным 0 пикселов, третий устанавливает отступ между границей ячейки и элементом, помещенным в нее, в 0 пикселов.

Другой пример – реализация меню с изменяющимся при наведении мышки видом.

При движении мыши у соответствующего компонента, попавшего в фокус мыши, "отгибается уголок". В данном случае "уголок" — это самостоятельная картинка. Все уголки реализованы в правой колонке таблицы. Для того чтобы гипертекстовая ссылка срабатывала по обеим картинкам (тексту и "уголку"), применяются одинаковые контейнеры A, охватывающие графические образы. В этом решении есть один недостаток: при переходе от текста к "уголку" последний "подмигивает". Картинки можно разместить и в одной ячейке таблицы, но тогда нужно задать ее ширину, иначе при изменении размеров окна браузера картинки могут "съехать". Чтобы убрать "подмигивание", необходимо сделать полноценные картинки замены.

"Подмигивание" происходит при переходе с одного элемента разметки контейнера на другой. При этом заново определяются свойства отображения элемента.

<html>

<head>

<title>UntitledDocument</title>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

<script  language="JavaScript">

       var i=0;

        function select(i)

        {

               var t ='document.evente'+i+'.src='+'"images/corner.gif";';

               eval(t);

        }

        function unselect(i)

        {

               var t ='document.evente'+i+'.src='+'"images/clear.gif";';

               eval(t);

        }

</script>

</head>

<body>

<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="center">

<TR>

        <TD><A HREF="" onMouseover="javascript:select(1)"

               onMouseout="javascript:unselect(1)">

               <IMG SRC="images/image9.gif" border="0"></A>

        </TD>

        <TD><A HREF="javascript:void(0);" onMouseover="javascript:select(1)"

               onMouseout="javascript:unselect(1)">

               <IMG NAME="evente1" SRC="images/clear.gif" border="0"></A>

        </TD>

</tr>

<tr>

        <TD><A HREF="" onMouseover="javascript:select(2)"

               onMouseout="javascript:unselect(2)">

               <IMG SRC="images/image10.gif" border="0"></A>

        </TD>

        <TD><A HREF="javascript:void(0);" onMouseover="javascript:select(2)"

               onMouseout="javascript:unselect(2)">

               <IMG NAME="evente2" SRC="images/clear.gif" border="0"></A>

        </TD>

</TR>

</TABLE>

</body>

</html>

**Задание**

1.                 Создать html документ, в котором с помощью JavaScript:

a.    Сгенерировать следующий текст:

Лабораторная работа №3

Студента группы (номер группы)

Фамилия Имя Отчество (в родительном падеже)

b.    На страницу добавить рисунок, который менял бы цветность при нажатии на кнопку бесконечное число раз (%-остаток от деления);

c.    Второй рисунок на странице должен быть псевдоанимацией, т.е. меняться последовательно до нажатия на кнопку, при повторном нажатии процесс должен возобновляться;

d.    Сформировать в верхней части страницы меню, при щелчках на пункты которого отображались бы информационные окна. При желании, можно организовать переходы на другие страницы;

e.    Сформировать вертикальное меню с графическим отображением наведения мыши. При желании можно не использовать графику.

2.                 Оформить отчет о проделанной работе.

**Лабораторная работа №4**

**Краткие теоретические сведения.**

Типы и структуры данных

Как и любой другой язык программирования, JavaScript поддерживает встроенные *типы и структуры данных*. Все их многообразие подразделяется на:

  литералы и переменные;

  массивы, функции и объекты.

При этом все они делятся на встроенные и определяемые программистом. Функции и объекты рассматриваются в разделах "Функции" и "Объекты". Поэтому здесь мы остановимся на литералах, переменных и массивах.

Литералы

Литералом называют данные, которые используются в программе непосредственно. При этом под данными понимаются числа или строки текста. Все они рассматриваются в JavaScript как элементарные *типы данных*. Приведем примеры литералов:

числовой литерал: 10

числовой литерал: 2.310

числовой литерал: 2.3e+2

строковый литерал: 'Это строковый литерал'

строковый литерал: "Это строковый литерал"

Литералы используются в операциях присваивания значений переменным или в операциях сравнения:

var a=10;

varstr = 'Строка';

if(x=='test') window.alert(x);

Два варианта строковых литералов необходимы для того, чтобы использовать вложенные строковые литералы. Вообще говоря, есть подозрение, что равноправие "..." и '...' мнимое. Если внимательно посмотреть на реализацию страниц программирования гипертекстовых ссылок (href.htm, path.htm и mouse.htm), можно заметить, что вместо прямого переназначения гипертекстовой ссылки литералом типа '...' там используется косвенное переназначение через функцию литералом "...":

...

function line(a)

{

...

window.document.main.document.links[4].href=

"javascript:data(0);void(0);";

...

}

...

<A HREF="javascript:line(0);void(0);">

<IMG SRC=image.gif BORDER=0>

</A>

вместо:

<A HREF="javascript:

window.document.main.document.links[4].href=

'javascript:data(0);void(0);';void(0);">

<IMG SRC=image.gif BORDER=0>

</A>

Это связано с особенностями реализации Netscape. Дело в том, что прямое переназначение неправильно отображает кириллицу в win32, а вот косвенное работает. Похоже, что "..." разрешает анализ информации внутри строкового литерала JavaScript-интерпретатором, а '...' — нет.

Если быть более точным, то следует сказать, что строка — это *объект*. У этого *объекта* существует великое множество методов. Строчный литерал и строчный *объек*т — далеко не одно и то же. При применении к строчным литералам методов строчных *объектов* происходит преобразование первых в последние.

Переменные

Переменные в JavaScript могут быть определены назначением или при помощи оператора var:

i=10;

var i;

var i=10;

var id = window.open();

var a = newArray();

Как видно из примеров, переменные могут принимать самые разные значения, при этом тип переменной определяется контекстом.

Массивы

Массивы делятся на встроенные (document.links[], document.images[],...) и определяемые пользователем (автором документа). Для массивов задано несколько методов:

join();

reverse();

sort();

и свойство length, которое позволяет получить число элементов массива. Это свойство активно используется в примерах данного раздела. В частности, при обсуждении метода join().

Для определения массива пользователя существует специальный конструктор:

a = new Array();

b = new Array(10);

c = newArray(10,"Это значение");

Пример использования:

<SCRIPT>

c = newArray(30,"Это значение");

</SCRIPT>

<FORM><INPUT SIZE=& {c[0];};

value=& {c[1];};

onFocus="this.blur();">

</FORM>

Как видно из этого примера, массив может состоять из разнородных элементов. Массивы не могут быть многомерными.

Для работы с массивами в JavaScript применяются методы join(), reverse(), sort(). Кроме того, массивы обладают свойством длины,length.

*Метод join()*

Метод join() позволяет объединить элементы массива в одну строку. Он является обратной *функцией* методу split(), который применяется к*объектам* типа STRING. Рассмотрим пример преобразования локального URL в URL схемы http:

window.location:

http://ya.ru/help/index.html

Выполнили:

b = window.location.href.split('/');

Получили массив b:

b[0]=http:

b[1]=

b[2]=intuit.ru

b[3]=help

b[4]=index.html

Заменили схему и вставили "host:port":

for(i=0;i<b.length;i++)

{

if(b[i]=="file:") b[i]="http:/";

if(b[i]=="c%7C") b[i]="remote.host.domain:80";

}

Получили массив b:

b[0]=http:

b[1]=

b[2]=ya.ru

b[3]=help

b[4]=index.html

Слили элементы массива b:

l=b.join("/");

Получили в результате:

http://ya.ru/help/index.html

Другой пример использования метода join() — замена символа в строке:

str = "document.img1.src=

'http://images/imagе1.gif';"

document.write(str);

Исходная строка:

document.img1.src='http://images/imagе1.gif';

Заменяем в строке все единицы на двойки:

b = str.split('1');

str = b.join('2');

Получаем следующий результат:

document.img2.src='http://images/imagе2.gif';

Последний пример показывает, что массив пользователя можно получить и без явного применения конструктора массива. Массив элементов строки получается просто как результат действия *функции* split().

*Метод reverse()*

Метод reverse() применяется для изменения на противоположный порядка элементов массива внутри массива. Предположим, массив натуральных чисел упорядочен по возрастанию:

a = newArray(1,2,3,4,5);

Упорядочим его по убыванию:

a.reverse();

a[0]=5

a[1]=4

a[2]=3

a[3]=2

a[4]=1

Можно, наверное, придумать и более внятный пример, но, к сожалению, на практике встречаться с этим методом мне приходилось нечасто.

*Метод sort()*

Как принято в современных интерпретируемых языках, например в Perl, метод sort() позволяет отсортировать элементы массива в соответствии с некоторой *функцией* сортировки, чье имя используется в качестве аргумента метода:

a = new Array(1,6,9,9,3,5);

function g(a,b)

{

if(a > b) return 1;

if(a < b) return -1;

if(a==b) return 0;

}

b = a.sort(g);

В результате выполнения этого кода получим массив следующего вида:

b[0]=1

b[1]=3

b[2]=5

b[3]=6

b[4]=9

b[5]=9

Возможность использования произвольной *функции* сортировки позволяет выполнять подробный анализ строковых *объектов*. Одним из таких примеров может служить анализ строки атрибута SRC контейнера IMG, если картинка подставляется скриптом, и сортировка полей формы по значениям:

document.image.src =

  "http://www.ya.ru:80/cgi-bin/

  image?x=10&y=20&z=15";

Выделим x и y. Затем отсортируем их:

str = "http://www.ya.ru:80/cgi-bin/

  image?x=10&y=20&z=15";

s = str.split('?');

s1 = s[1].split('&');

s2 = s1.sort(g);

for(i=0;i<s2.length;i++)

  document.write("s2["+i+"]=

  '"+s2[i]+"'<br>");

s2[0]='x=10'

s2[1]='z=15'

s2[2]='y=20'

Аналогичные манипуляции можно проделать с любым массивом. Если не указывать *функции* в аргументе метода сортировки, то элементы массива сортируются в лексикографическом порядке. Это значит, что они сначала преобразуются в строки и только потом сортируются.

Функции

Язык программирования не может обойтись без механизма многократного использования кода программы. Такой механизм обеспечивается *процедурами*или *функциями*. В JavaScript *функция* выступает в качестве одного из основных *типов данных*. Одновременно с этим в JavaScript определен *объект*  Function.

В общем случае любой *объект* JavaScript определяется через *функцию*. Для создания *объекта* используется конструктор, который в свою очередь вводится через Function. Таким образом, с *функциями* в JavaScript связаны следующие ключевые вопросы:

* *функция* — *тип данных*;
* *функция* — *объект*;
* конструкторы *объектов*.

Именно эти вопросы мы и рассмотрим в данном разделе.

*Функция — тип данных*

Определяют *функцию* при помощи ключевого слова function:

functionf\_name(arg1,arg2,...)

{

/\* functionbody \*/

}

Здесь следует обратить внимание на следующие моменты. Во-первых, function определяет переменную f\_name. Эта переменная имеет тип "function":

document.write("Тип переменной f\_name:"+

                typeof(f\_name));

Тип переменной f\_name: function. Во-вторых, этой переменной присваивается значение:

document.write("Значение i:"+i.valueOf());

document.write("Значениеf\_name:"+

                f\_name.valueOf());

Значение переменной f\_name: 10. Значение переменной f\_name:functionf\_name(a) { if(a>=0) return true; else return false; }. В данном случае метод valueOf() применяется как к числовой переменной i, так и к f\_name. По этой причине *функции* можно назначить синоним путем присваивания ее значения другой переменной:

functionf\_name(a)

{

if(a>=0) return true; else return false;

}

document.write("Значение переменной f\_name:"+

                f\_name(1)+"");

b = f\_name;

document.write("Значение переменной b:"+

                b(1)+"");

Значение переменной f\_name:true

Значение переменной b:true

Очевидно, что если *функцию* можно присвоить переменной, то ее можно передать и в качестве аргумента другой *функции*. Все это усиливается при использовании *функции* eval(), которая позволяет реализовать отложенное исполнение JavaScript-кода. Отложенное исполнение — это возможность изменения программы по ходу ее исполнения. Типичным использованием eval() является сокращение кода за счет генерации однотипных строк:

for(i=0;i<5;i++)

{

eval("document.write('test"+i+"<br>')");

}

Результат исполнения кода:

test0

test1

test2

test3

test4

При непосредственном кодировании пришлось бы написать пять строк кода. Данный подход использовался в разделе "Изменение картинки" для инициирования массивов картинок, имена которых построены по принципу индексации ячеек таблицы.

*Функция — объект*

У любого *типа данных* JavaScript существует *объектовая* "обертка" — Wrapper, которая позволяет применять методы *типов данных* к переменным и литералам, а также получать значения их свойств. Например, длина строки символов определяется свойством length. Аналогичная "обертка" есть и у*функций* — *объект*   Function.

Например, увидеть значение *функции* можно не только при помощи метода valueOf(), но и используя метод toString():

functionf\_name(x,y)

{

return x-y;

}

document.write(f\_name.toString()+"<br>");

Результат распечатки:

functionf\_name(x,y) { return x-y; }

Свойства *функции* доступны для программиста только тогда, когда они вызываются внутри *функции*. При этом обычно программисты имеют дело с массивом аргументов *функции* (arguments[]), его длиной (length), именем *функции*, вызвавшей данную *функцию* (caller)и *прототипом* (prototype).

Рассмотрим пример использования списка аргументов *функции* и его длину:

functionmy\_sort()

{

a = new Array(my\_sort.arguments.length);

for(i=0;i<my\_sort.arguments.length;i++)

a[i] = my\_sort.arguments[i];

b = a.sort();

return b;

}

b = my\_sort(9,5,7,3,2)

for(i=0;i<b.length;i++)

document.write("b["+i+"]="+b[i]+"<br>");

Результат исполнения:

b[0]=2

b[1]=3

b[2]=5

b[3]=7

b[4]=9

Если *функция* может быть вызвана из других *функций*, то в этом случае используется свойство caller:

function slave()

{

document.write(slave.caller+"");

returnslave.caller;

}

function master1()

{

slave();

}

function master2()

{

slave();

}

...

master1();

master2();

Результат исполнения двух последних строк:

function master1() { slave(); }

function master2() { slave(); }

Еще одним свойством *объекта*   Function является prototype, но это общее свойство всех *объектов*, поэтому и обсуждать его мы будем в контексте*типа данных*   Object. Упомянем только о конструкторе *объекта*   Function:

f = newFunction(arg\_1,...,arg\_n, body)

Здесь f — это *объект* класса Function. Его можно использовать и как обычную *функцию*. Конструктор используют для получения безымянных *функций*, которые назначают или переопределяют методы *объектов*. Здесь мы вплотную подошли к вопросу конструирования *объектов*. Дело в том, что переменные внутри *функции* можно рассматривать в качестве ее свойств, а *функции* – в качестве методов:

function Rectangle(a,b,c,d)

{

this.x0 = a;

this.y0 = b;

this.x1 = c;

this.y1 = d;

this.area = new Function(

  "returnMath.abs(this.x0-this.x1)\*

   Math.abs(this.y0-this.y1)");

this.perimeter = new Function(

  "return (Math.abs(this.x0-this.x1)+

   Math.abs(this.y0-this.y1))\*2");

}

c = new Rectangle(0,0,100,100);

document.write(c.area());

Результат исполнения:

10000

Обратите внимание еще на одну особенность — ключевое слово this. Оно позволяет сослаться на текущий *объект*, в рамках которого происходит исполнение JavaScript-кода. В данном случае это *объект* c класса Rectangle.

Объекты

***Объект*** — это главный *тип данных* JavaScript. Любой другой *тип данных* имеет *объектовую* "обертку" — Wrapper. Это означает, что прежде чем можно будет получить доступ к значению переменной того или иного типа, происходит конвертирование переменной в *объект*, и только после этого выполняются действия над значением. *Тип данных*   Object сам определяет *объекты*.

В данном разделе мы остановимся на трех основных моментах:

* понятие *объекта*;
* *прототип* *объекта*;
* методы *объекта*   Object.

Мы не будем очень подробно вникать во все эти моменты, так как при программировании на стороне браузера чаще всего обходятся встроенными средствами JavaScript. Но поскольку все эти средства — *объекты*, нам нужно понимать, с чем мы имеем дело.

*Понятие объекта*

Сначала рассмотрим пример произвольного, определенного пользователем *объекта*, потом выясним, что же это такое:

function Rectangle(a,b,c,d)

{

this.x0 = a;

this.y0 = b;

this.x1 = c;

this.y1 = d;

this.area = new Function(

  "returnMath.abs(this.x0-this.x1)\*

   Math.abs(this.y0-this.y1)");

this.perimeter = new Function(

  "return (Math.abs(this.x0-this.x1) +

   Math.abs(this.y0-this.y1))\*2");

}

c = new Rectangle(0,0,100,100);

document.write(c.area());

Результат исполнения:

10000

Этот же пример использовался в разделе "*Функции*" для иллюстрации применения конструктора *функции*. Здесь мы рассмотрим его в более общем контексте.

*Функция*   rectangle() — это конструктор *объекта* класса Rectangle, который определяется пользователем. Конструктор позволяет создать реальный*объект* данного класса. Ведь *функция* — это не более чем описание некоторых действий. Для того чтобы эти действия были выполнены, необходимо передать *функции* управление. В нашем примере это делается при помощи оператора new . Он вызывает *функцию* и тем самым генерирует реальный*объект*.

Создается четыре переменных: x0, y0, x1, y1 — это свойства *объекта*   Rectangle. К ним можно получить доступ только в контексте *объекта* данного класса, например:

up\_left\_x = c.x0;

up\_left\_y = c.y0;

Кроме свойств мы определили внутри конструктора два *объекта* типа Function(), применив встроенные конструкторы языка JavaScript, — area иperimeter. Это методы *объекта* данного класса. Вызвать эти *функции* можно только в контексте *объекта* класса Rectangle:

sq = c.area();

length = c.perimeter();

Таким образом, *объект* — это совокупность свойств и методов, доступ к которым можно получить, только создав при помощи конструктора *объект*данного класса и использовав его контекст.

На практике довольно редко приходится иметь дело с *объектами*, созданными программистом. Дело в том, что *объект* создается *функцией*-конструктором, которая определяется на конкретной странице и, следовательно, все, что создается в рамках данной страницы, не может быть унаследовано другими страницами. Нужны очень веские основания, чтобы автор Web-узла занялся разработкой библиотеки классов *объектов*пользователя. Гораздо проще писать *функции* для каждой страницы.

*Прототип*

Обычно мы имеем дело со встроенными *объектами* JavaScript. Собственно, все, что изложено в других разделах курса — это обращение к свойствам и методам встроенных *объектов*. В этом смысле интересно свойство *объектов*, которое носит название prototype. Прототип — это другое название конструктора *объекта* конкретного класса. Например, если мы хотим добавить метод к *объекту* класса String:

String.prototype.out = new Function("a",

    "a.write(this)");

...

"Привет!".out(document);

Результат исполнения:

Привет!

Для объявления нового метода для *объектов* класса String мы применили конструктор Function. Есть один существенный нюанс: новыми методами и свойствами будут обладать только те *объекты*, которые порождаются после изменения прототипа *объекта*. Все встроенные *объекты* создаются до того, как JavaScript-программа получит управление, что существенно ограничивает применение свойства prototype.

*Методы объекта Object*

Object — это *объект* и, следовательно, у него могут быть методы. Таких методов мы рассмотрим три: toString(), valueOf() и assign().

Метод toString() осуществляет преобразование *объекта* в строку символов. Он используется в JavaScript-программах повсеместно, но неявно. Например, при выводе числа или строковых *объектов*. Интересно применение toString() к *функциям*:

document.write(prot.toString()+"<BR>");

Результатисполнения:

function prot() {

a = this.src.split(':');

protocol = a[0]+":";

returnprotocol; }

Здесь мы используем *функцию*   prot() из примера с прототипом. Если распечатать таким же образом *объект*   Image, то получим следующее:

картинка:[object]

Таким образом, далеко не всегда метод toString() возвращает строковый эквивалент содержания *объекта*. Он может просто вернуть его тип. InternetЕхplorer при этом возвращает "Object", в то время как NetscapeNavigator — "objectImage".

Аналогично ведет себя и метод valueOf(). Этот метод позволяет получить значение *объекта*. В большинстве случаев он работает подобно методуtoString(), особенно если нужно выводить значение на страницу:

document.write(prot.valueOf()+"<BR>");

Результатисполнения:

function prot() {

a = this.src.split(':');

protocol = a[0]+":";

returnprotocol; }

Как видим, результат тот же, что и в методе toString().

В отличие от двух предыдущих методов, assign() позволяет не прочитать, а переназначить свойства и методы *объекта*. Данный метод используется в контексте присваивания *объекту* некоторого значения:

object = value; <=>object.assign(value);

Рассмотрим пример:

c = newImage();

c.src = "image1.gif";

b =new Image();

b.src = "image2.gif";

Image.prototype.assign =

  new Function("a","this.src = a.src;");

...

<A HREF='javascript:window.document.main.

document.i2.assign(c);

void(0);'>Покрасить картинку</A>

В данном примере мы применяем метод assign() не внутри JavaScript-кода, а в обычной HTML-разметке. При этом перед использованием метода мы переопределили его своей *функцией*. Пользователей InternetExplorer следует предупредить, что здесь мы их немножко обманываем, так как переназначить прототип Image в InternetExplorer мы в данном случае не можем.

Операторы языка

В этом разделе будут рассмотрены операторы JavaScript. Основное внимание при этом мы уделим операторам декларирования и управления потоком вычислений. Без них не может быть написана ни одна JavaScript-программа.

Общий перечень этих операторов выглядит следующим образом:

var;

{...};

if;

while;

for;

for ... in;

break;

continue;

return.

*var*

Оператор var служит для объявления переменной. При этом переменная может принимать значения любого из разрешенных типов данных. На практике довольно часто обходятся без явного использования var. Переменная соответствующего типа создается путем простого присваивания:

var a;

var a=10;

var a = new Array();

var a = newImage();

Все перечисленные выше примеры использования var верны и могут быть применены в JavaScript-программе. Область действия переменной определяется блоком (составным оператором), в котором используется переменная. Максимальная область действия переменной — страница.

*{...}*

Фигурные скобки определяют составной оператор JavaScript — блок. Они одновременно ограничивают область действия переменных, которые определены внутри этих скобок. За пределами блока переменные не видны:

{

var i=0;

}

Основное назначение блока — определение тела цикла и тела условного оператора.

*if*

Условный оператор применяется для ветвления программы по некоторому логическому условию. Общий синтаксис:

if (логическое выражение) оператор1;

[else оператор2;]

Логическое выражение — это выражение, которое принимает значение true или false. Если оно равно true, то оператор 1 исполняется. В квадратных скобках необязательная составляющая оператора if — альтернатива основной ветви вычислений:

if (navigator.appName=="Netscape")

{

window.location.href=

"http://ya.ru/netscape.htm";

}

else

{

window.location.href=

"http://ya.ru/explorer.htm";

}

Примеры использования условного оператора можно найти, например, в разделе "Тип браузера".

*while*

Оператор while определяет цикл. Определяется он в общем случае следующим образом:

While (логическое выражение)

оператор;

Оператор, в том числе и составной, — тело цикла. Тело исполняется до тех пор, пока верно логическое условие:

while (flag==0)

{

id=setTimeout ("test();",500);

}

Обычно цикл этого типа применяют при выполнении периодических действий до некоторого события.

*for*

Оператор for — это еще один оператор цикла. В общем случае он имеет вид:

for (инициализация переменных цикла;

     условие; модификация переменных цикла)

оператор;

Оператор в теле цикла может быть блоком. Рассмотрим типичный пример использования этого оператора:

for(i=0;i<document.links.length;i++)

{

document.write(document.links[i].href+"<BR>");

}

http://ya.ru/help/index.html

http://ya.ru/help/shop.html#choice

http://ya.ru/help/payment.html

Подобные примеры разбросаны по всем разделам курса.

*for ... in*

Данный оператор позволяет "пробежаться" по свойствам *объекта*. Рассмотримпример:

for(v in window.document)

{

document.write(v+"<BR>");

}

*break*

Оператор break позволяет досрочно покинуть тело цикла. Распечатаемтолько title документа:

for(v in window.document)

if(v=="title")

{

document.write(v+":"+eval('document.'+v)+"

");

break;

}

Результатисполнения:

title:Web-engineering

(Introduction to the JavaScript. Operators.).

В пример распечатки свойств *объекта*   document мы вставили break при просмотре свойства title и получили искомый результат.

*continue*

Того же результата, что и при использовании break, можно было бы достичь при помощи оператора continue:

for(v in window.document)

{

if(v!="title") continue;

document.write(v+":"+eval('document.'+v));

break;

}

Результат исполнения:

title:Web-engineering

(Introduction to the JavaScript. Operators.)

Этот оператор позволяет пропустить часть тела цикла (от оператора до конца тела) и перейти к новой итерации. Таким образом мы просто пропускаем все свойства до title и после этого выходим из цикла.

*return*

Оператор return используют для возврата значения из *функции* или обработчика события (см. разделы "Поле статуса", "*Обмен данными*"). Рассмотримпример:

<FORM>

<INPUT TYPE=submit VALUE=Submit

onClick="return false;">

</FORM>

В данном примере return используется для маскирования передачи данных на сервер.

**Задание**

1.                 Создать html документ, в котором с помощью JavaScript:

a.    Сгенерировать следующий текст:

Лабораторная работа №4

Студента группы (номер группы)

Фамилия Имя Отчество (в родительном падеже)

b.    Создать процедуру ввода массива;

c.    Отсортировать массив;

d.    Если первый элемент отсортированного массива меньше некоторого вводимого N, то найти элемент, который больше или равен N и сформировать новый укороченный массив, начиная с N;

e.    Вывести полученный массив на экран.

f.     Записать url страницы в виде массива и вывести поэлементно на экран

Примечание: все задания нужно оформить в виде функций и поместить в отдельный скриптовый файл, подключаемый кИнтернет – странице.

2.                 Оформить отчет о проделанной работе.

**РАЗРАБОТКА WEB - ПРЕДСТАВИТЕЛЬСТВ.**

**Лабораторные работы**

                                                                                                                      Автор: Градусов Д.А.

**Содержание:**

* [Указания к отчетному заданию по лабораторным работам](http://www.dec.vlsu.ru:81/pluginfile.php/164085/mod_resource/content/0/P1000/WEBlab/zach.htm)

|  |  |  |
| --- | --- | --- |
| 1. | 1. Создать сайт состоящий из 5 связанных страниц(разделов).Пример сайта(Разработайте Web-страницы, рассказывающие о вашем классе. На головной странице поместите рассказ о классе, классном руководителе. Рассказ об учениках разместите на отдельных Web-страницах. Укажите ссылки на страницы учеников с головной страницы. Не забудьте установить ссылки возврата с Web-страниц учеников на головную страницу.) Можно сайт по любой тематике. присылайте сайты в заархивированном виде (zip). |  |
|  | Чтобы вложить файл в задание:   * 1. Сохраните результаты. Убедитесь, что вам известно расположение сохраненного файла.   2. Нажмите кнопку **Обзор**.   3. Найдите и выберите сохраненный файл.   4. Нажмите кнопку **Открыть**, чтобы вложить файл в задание. |  |