7 ЛАБОРАТОРНАЯ РАБОТА «ПРОЕКТИРОВАНИЕ СКРИПТОВ»

В процессе выполнения лабораторной работы студенты выполняют проектирование скриптов (JavaScript) обработки данных электронных форм.

Клиентский скрипт - это программа, которая может сопровождать документ HTML или непосредственно быть внедренной в него. Эта программа выполняется на клиентской машине при загрузке документа или в другое время, например, когда активизируется ссылка. Поддержка скриптов в HTML не зависит от языка скрипта.

Для лабораторной работы выполняется стандартный пример использования скриптов - обработка данных электронной формы, заполненной пользователем:

- проверка правильности введенных данных;
- выдача на экран сообщений;
- преобразование данных;
- запись данных в файл;
- отправка ответа пользователю;
- и другие необходимые операции.

В качестве примера такого использования скрипта проектируется фрагмент электронного интерактивного методического материала для изучения свойств элементов языка HTML.

7.1 JavaScript

Язык JavaScript - это объектно-ориентированный язык, предназначенный для создания приложений в Интернет. Язык JavaScript является системно - независимым и поддерживается браузерами Netscape Navigator (NN), начиная с версии 2.0 и Microsoft Internet Explorer (IE). Код скрипта (сценарий), написанный на языке JavaScript, включаются в состав HTML документа и, фактически, этот язык может считаться расширением HTML. Поэтому для написания скриптов, как и HTML – документов, достаточно текстового редактора.

Основными понятиями этого языка являются:

- объект;
- методы и атрибуты объектов;
- функции.

Используемый в составе HTML – документа скрипт служит для увеличения функциональности и возможностей взаимодействия с пользователями.

Скрипты предлагают авторам средства усиления интерактивности документов HTML. Например:

- скрипты могут оцениваться во время загрузки документа и динамически изменять содержимое документа;

- скрипты могут использоваться в форме для обработки вводимых данных. Дизайнеры могут динамически заполнять поля формы в зависимости от значений других полей. Они могут проверять, попадают ли введенные данные в предопределенный диапазон значений, соответствие полей и т.д;
- скрипты могут включаться событиями, оказывающими влияние на документ, например, загрузкой, выгрузкой, фокусом элемента, перемещением мыши и т.д;
- скрипты могут связываться с управляющими элементами формы (например, с кнопками) для представления элементов пользовательского интерфейса.

7.1.1 Определение каждого скрипта в HTML - документе производится при помощи элемента SCRIPT, с необязательным атрибутом language. Например, следующий фрагмент:

```
<BODY>
<SCRIPT type="text/javascript" language="JavaScript">
document.write("Наш первый пример на JavaScript");
</SCRIPT>
</BODY>
```

Выводит на экран строку «Наш первый пример на JavaScript».

7.1.2 Размещение сценария скрипта в заголовке документа HEAD, обеспечивает загрузку скрипта до отображения браузером тела страницы. Следующий пример демонстрирует правильное определение скрипта:

```
<HTML>
<HEAD>
<!-- Заключаем сценарий в теги комментария
<SCRIPT language="JavaScript">
        Код сценария
-->
</SCRIPT>
</HEAD>
<BODY>
        Код  HTML - документа
</BODY>
</HTML>
```

Регистр, которым написаны буквы, в JavaScript имеет значение. Авторы могут указывать язык скрипта по умолчанию для всех скриптов в документе, включив следующее объявление META:

```
<META http-equiv="Content-Script-Type" content="type">
```

где "type" - тип содержимого, именующий язык скрипта. Примерами значений являются "text/tcl", "text/javascript", "text/vbscript".

7.1.3 Объектная модель документа основана на понятии объекта. Объекты представляют собой блоки, из которых строится JavaScript. Объектами являются само текущее окно браузера «Window», текущий документ обозначается как объект с именем «Document». Объектами являются большинство элементов, составляющих документ:

- тело документа;
- ссылки;
- элементы контейнеры;
- формы;
- управляющие элементы форм и т.д.

Они применяются для возвращения значений и изменения состояния форм, страниц, браузера и определенных программистом переменных.

Объекты могут создаваться программистом во время выполнения скрипта при помощи функции-конструктора. Например, следующая функция должна создавать новый экземпляр объекта «Dom»:

```
function Dom(bedrooms, bathrooms, floors, squareFeet)
{
this.bedrooms - bedrooms;
this.bathrooms - bathrooms;
this.floors - floors;
this.squareFeet = squareFeet
}
```

Параметрами объекта будут являться количество спален, ванных, залов и площадь дома.

Теперь, когда объект определен, его экземпляр можно создать при помощи оператора new.

```
Dom1 = new Dom (2, 1, 1, 1700);
```

Каждый объект может описываться набором свойств (атрибутов). Так тело документа BODY описывается цветом фона bgcolor. Следующий пример демонстрирует динамическое изменение цвета фона документа:

```
<HTML>
<HEAD>
<TITLE>Цвет фона</TITLE>
</HEAD>
<BODY bgcolor="ffffff">
<FORM style="text-align: center">
<INPUT type=button value="красный" onClick="document.bgColor='ff0000'">
<INPUT type=button value="желтый" onClick="document.bgColor='ffff00'">
<INPUT type=button value="синий" onClick="document.bgColor='0000ff'">
</FORM>
<H2 align=center>Не забудьте закрыть окно!</H2>
```

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

Событие наступает при выполнении пользователем нажатия клавиши - событие «onClick».

Функции, ассоциированные с объектом, называются методами объекта. Функция выполняет определенные действия, например, document.write(parameter) выводит parameter в текущий документ.

Каждый объект имеет свой набор атрибутов и функций. Внешне функция отличается от объекта наличием круглых скобок ().

7.1.4 Кроме функций, принадлежащих объектам, Вы можете создавать и свои функции. Обычно функция создается при необходимости многократного вызова одной и той же последовательности команд. Следующий пример определяет функцию (clock_form()), запускающуюся автоматически через определенный интервал времени, а не по действию пользователя. Таким образом, в окне браузера реализуется работа часов:

```
<HTML>
<HEAD>
<TITLE>Часы</TITLE>
<SCRIPT language="JavaScript">
function clock_form()
      {
     day=new Date()
     clock_f=day.getHours()+":"+day.getMinutes()+":"+day.getSeconds()
     document.form.f_clock.value=clock_f
      id=setTimeout("clock_form()",100)
     }
</SCRIPT>
</HEAD>
<BODY onLoad="clock_form()">
<FORM name=form>
<INPUT name=f_clock maxlength=8 size=8>
</FORM>
<H2>Не забудьте закрыть окно!</H2>
</BODY></HTML>
```

7.1.5 Иерархия объектов, используемых в языке, очень похожа на объектную модель среды разработки Delphi. Корневым элементом – вершиной дерева объектов является объект «Window», представляющий открытое окно браузера. При определении объекта, если не указан корневой элемент, всегда подразумевается открытое окно браузера. Этот объект является родительским по отношению ко всем остальным объектам страницы - document; location, history.

Объект document в свою очередь содержит все вложенные объекты – заголовок, ссылки, графические элементы, фреймы и формы. Каждый из объектов определяется набором атрибутов, которые доступны для чтения и изменения через обращения к их именам.

Если вложенных элементов больше одного, для их определения создается индексный список. Нумерация осуществляется по порядку следования элементов, начиная с нулевого элемента.

Так обращение к двум управляющим элементам ввода текста в форме производится следующим образом:

```
document.form.text[0].value
document.form.text[1].value
```

Количество вложенных элементов определяется через свойство объекта «length».

7.2 Синтаксис и команды

7.2.1 В JavaScript существуют следующие типы данных:

- строка (String) - последовательность символов, заключенных в кавычки;
- число (Numbers) - целые числа и десятичные дроби;
- логические (Boolean) – принимающие значения True или False;
- значение null - отсутствие данных.

Также JavaScript поддерживает специальные символы, как и язык программирования C++:

- \n - Новая строка
- \t - Табуляция
- \f - Новая страница
- \b - Забой
- \r - Возврат каретки

Вы можете также избежать написания других символов, поставив перед каждым из них косую черту (\). Тем самым вы запретите браузеру их интерпретировать. Чаще всего это используется для кавычек и обратной косой черты или для включения символа с использованием его восьмеричного значения.

```
document.write("Здесь показаны \"кавычки\" внутри строки");
document.write("Это обратная косая черта \\");
document.write("Это символ пробела \040");
```

7.2.2 Переменные JavaScript не имеют ярко выраженного типа. Вместо этого каждая переменная может содержать значения разных типов. Преобразование из одного типа в другой делается автоматически, как показано в следующем примере:

```
x = 55; // x - переменная 55 число;
```

y = "55"; // y - строковая переменная "55";
y = '55'; // альтернативное использование одиночных кавычек;
z = 1 + y.

Если y - строковая переменная, то она будет автоматически преобразована в ближайшее целое значение так, чтобы 1 могла быть прибавлена к ней.

Число 55 выводится на экран. Если x - целое, а не строковая переменная, JavaScript выполнит необходимые преобразования - document.write(x).

Задание значений числовых переменных выполняется по одному из следующих форматов:

- n = 3.14159265; // назначение действительного (дробного) числа;
- n = 0546; // числа, начинающиеся с нуля, - восьмеричные;
- n = 0xFFEC; // числа, начинающиеся с 0x, - шестнадцатеричные;
- n = 2.71862E-5; // использование экспоненциальной нотации.

7.2.3 Имена переменных должны начинаться с буквы или подчеркивания. После первого символа переменная может содержать любую комбинацию из букв, подчеркивания и цифр.

Язык JavaScript чувствителен к регистру, так что «Var17» и «var17» будут рассматриваться как разные переменные.

Перед использованием переменных необязательно их объявлять. Тем не менее, вы можете задать переменную явно с помощью ключевого слова «var». var x = "55".

7.2.4 Операторы присваивания. Для присваивания значений в JavaScript используются операторы, список которых приведен в таблице.

Таблица 7.1 – Операторы присваивания

Оператор	**Пример**	**Результат**
=	x = y	x равно y
+ =	x += y	x равно x + y
- =	x -= y	x равно x минус y
* =	x *= y	x равно x умножить на y
/ =	x /= y	x равно x разделить на y
% =	x %= y	x равно остатку x по модулю y

Каждый из этих операторов присваивает значение справа переменной слева, как показано в следующем примере:

x = 100; y = 10; x += y.

Значение x после выполнения всех операторов равно 110.

7.2.5 Операторы сравнения, используемые в JavaScript, приведены в следующей таблице.

Таблица 7.2– Операторы сравнения

Оператор	Результат
= =	Равно
! =	Не равно
>	Больше, чем
<	Меньше, чем
> =	Больше или равно
< =	Меньше или равно

Например, проверка на вхождение в заданный интервал переменной wd:

if ((wd>0) && (wd<=100000)) ttab.write(" width="+wd+wdv)

7.2.6 Операторы выполнения математических и логических действий приведены в таблице 7.3.

Таблица 7.3 – Математические операторы

Оператор	Результат
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления по модулю
++	Увеличение на 1
--	Уменьшение на 1
& или AND	Отрицание
| или OR	Поразрядное логическое И
^ или XOR	Поразрядное логическое ИЛИ
<<	Сложение по модулю 2

>>	Сдвиг на один разряд влево
>>>	Сдвиг на один разряд вправо
&&	Логическое И
||	Логическое ИЛИ
!	НЕ
// комментарий	Коментарий, занимающий одну строчку
/* комментарий несколько строк */	Комментарий занимающий несколько строк

7.2.7 Операторы, выполняемые над строковыми переменными, допускают объединение двух или нескольких строк или сравнение содержимого.

Таблица 7.4 – Строковые операторы

Оператор	**Результат**
+	Объединяет строки, то есть "abcd" + "efg" = "abcdefg"
>	Сравнивает строки путем сравнения их кодов ASCII, начиная с левого конца. Так, "DEF" больше, чем "ABC" или "DEE", но меньше, чем "abc".
> =	
<	
< =	

7.3 Объекты и функции JavaScript

7.3.1 Встроенные объекты. JavaScript предлагает для работы с различными типами данных ряд встроенных объектов, каждый из которых имеет свой собственный набор свойств и методов, доступных в коде JavaScript.

7.3.1.1 Объект «Array».

Объект Array определяет метод создания массивов и работы с ними. Для создания нового массива используется одна из следующих конструкций:

```
cast = Array(); // создание нового массива;
cast = Array(10); // создание массива из 10 элементов;
// создание массива и одновременно заполнение его значениями
cast = Array("Мурка", "Васька", "Толстопуз", "Барсик").
```

Для работы с массивами определены следующие функции:

- length - целое число с возможностью изменения, указывающее количество элементов массива;
- join([строка]) - возвращает строку, содержащую все элементы массива, строка в скобках - необязательный разделитель;
- reverse() - изменяет на обратный порядок следования элементов;
- sort([функция]) - сортирует массив. В случае указания функции в скобках - по результатам ее вычисления.

Более ранние версии JavaScript не имели явно выраженной конструкции массивов. Однако объектные механизмы JavaScript и там позволяют легко создавать массивы:

```
function MakeArray(n) {
  this.length = n;
  for (var i =1; i <= n; i ++)
    this(1) = 0;
  return this }
```

С помощью этой функции, включенной в ваш сценарий, можно создать массив:

```
cast = new MakeArray(20);
```

После этого вы можете его заполнить значениями:

```
cast[1] = "Мурка";
cast[2] = "Васька";
cast[3] = "Толстопуз";
cast[4] = "Барсик";
```

7.3.1.2 Объект «Boolean».

Объект Boolean, используется для хранения простых значений типа да/нет**,** true/false. Для создания нового объекта этого типа используется такой синтаксис:

```
MyAnsver = new Boolean([значение]);
```

Если значение равно 0, null, пропущено или является пустой строкой, новый объект будет иметь начальное значение false. Все другие значения этого аргумента, включая строку "false", приведут к созданию объекта, имеющего значение true. Объект имеет два метода:

- toString() - возвращает значение объекта как строку "true" или "false";
- valueOf() - возвращает численное значение объекта для использования в вычислениях.

7.3.1.3 Объект «Date».

Объект Date поддерживает метод для работы с датами и временем в JavaScript. Новые экземпляры объектов создаются так:

```
newDateObject = new Date([значение]);
```

Значение - необязательные установки даты и времени для нового объекта. Если они не заданны, используются текущие дата и время. Значение может иметь такой вид:

- миллисекунды - начиная с полуночи по Гринвичу 1 января 1970 года;
- год, месяц, день (например, 1997, 0, 27 - 27 января 1997);
- год, месяц, день, часы, минуты, секунды;
- месяц, день, год, часы:минуты:секунды (September 23, 1997, 08:25:30).

В примере пункта 1.1.4, задание идущих часов в окне браузера задается набором методов:

```
day=new Date()
clock_f=day.getHours()+":"+day.getMinutes()+":"+day.getSeconds()
```

Полный набор методов для работы с временными параметрами приведен в таблице:

Таблица 7.5 – Методы объекта

Метод	Описание
getDate()	Возвращает число месяца как целое число от 1 ло 31
getDay()	Возвращает день недели как целое число от 0 (воскресенье) до 6 (суббота)
getHours()	Возвращает часы как целое от 0 до 23
getMinutes()	Возвращает минуты как целое от 0 до 59
getSeconds()	Возвращает месяц как целое от 0 (январь) до 11 (декабрь)
getTime()	Возвращает секунды как целое от 0 до 59
getTimeZoneOffset()	Возвращает количество миллисекунд между 1 января 1970 года в 00:00:00 по Гринвичу и текущим объектом Date как целое число.
getYear()	Возвращает разницу в минутах между местным и гринвичским временем как целое число
parse (значение даты)	Возвращает год без первых двух разрядов как целое число.
setDate(значение)	Возвращает количество миллисекунд между 1 января 1970 года в 00:00:00 по Гринвичу и значением даты как целое число.
setHours(значение)	Устанавливает день с помощью целого числа значение от 1 до 31
setMinutes(значение)	Устанавливает часы с помощью целого числа значение от 0 до 23

setMounth(значение)	Устанавливает минуты с помощью целого числа значение от 0 до 59
setSeconds(значение)	Устанавливает секунды с помощью целого числа значение от 0 до 59
setTime(значение)	Устанавливает значение объекта Date с помощью целого числа значение, которое отражает количество миллисекунд между 1 января 1970 года в 00:00:00 по Гринвичу
setYear(значение)	Устанавливает часы с помощью целого числа значения большего 1900
toGMTString()	Преобразует данные местного времени во время по Гринвичу и возвращает как строку
toLocaleString()	Преобразует время по Гринвичу в данные местного времени и возвращает как строку
UTC(год, месяц, день [,часы] [,минуты] [,секунды])	Возвращает количество миллисекунд между 1 января 1970 года в 00:00:00 по Гринвичу и текущим объектом Date как целое число.

7.3.1.4 Объект Function предоставляет механизм для компиляции кода JavaScript в виде функции. Новая функция создается конструктором:

functionName = new Function(аргумент1, аргумент2, ... код функции);

где аргумент1, аргумент2 и т.д. - аргументы для создаваемого объекта, а код функции - строка, содержащая тело функции. Это может быть несколько выражений JavaScript, разделенных точками с запятой.
Объект функции описывается следующими свойствами:

- arguments[ ] - ссылка на массив Arguments, содержащий аргументы вызванной функции;
- caller - определяет функцию, вызываемую объекта Function;
- prototype - предоставляет способ для добавления свойств объекту.

7.3.1.5 Объект «Arguments».
Объект Arguments - список (массив) аргументов объекта Function имеет единственное свойство length, целое число, определяющее количество аргументов, необходимых вызванной функции.

7.3.1.6 Объект «Math» представляет набор свойств и методов для работы с математическими константами и функциями. Для его использования нужна сначала ссылка на объект Math а зетем - на требуемые метод или свойство:

```
MyArea = Math.PI * MyRadius * MyRadius;
MyResult = Math.floor(MyNumber);
```

В математических выражениях допускается применение следующих констант:

E	постоянная Эйлера - экспонента
LN10	Значение натурального логарифма числа 10
LN2	Значение натурального логарифма числа 2
LOG10E	Значение десятичного логарифма экспоненты
LOG2E	Значение двоичного логарифма экспоненты
PI	Значение постоянной Пи = 3.141592654 ...
SQRT	Значение квадратного корня из 2

Доступные математические функции приведены в следующей таблице:

Таблица 7.6 – Математические функции

Свойство	Описание
abs(число)	Возвращает модуль числа
acos(число)	Возвращает арккосинус числа
asin(число)	Возвращает арксинус числа
atan(число)	Возвращает аркиангенс числа
atan2(x, y)	Возвращает угол в полярных координатах точки с координатами x, y от оси x
ceil(число)	Округляет *число* вверх до ближайшего целого
cos(число)	Возвращает косинус числа
exp(число)	Возвращает экспоненту в степени
floor(число)	Округляет вниз до ближайшего целого
log(число)	Возвращает натуральный логарифм числа
max(число1, число2)	Возвращает большее из чисел
min(число1, число2)	Возвращает меньшее из чисел
pow(число1, число2)	Возвращает *число1* в степени *число2*
random()	Возвращает случайное число между 0 и 1
round(число)	Округляет *число* до ближайшего целого
sin(число)	Возвращает синус числа *число*
sqrt(число)	Возвращает квадратный корень из числа *число*

tan(число)	Возвращает тангенс числа *число*

7.3.1.7 Объект «Number» предоставляет набор свойств, полезных при работе с числами:

- MAX_VALUE - максимальное численное значение (~1.79E+308);
- MIN_VALUE - минимальное численное значение (~2.22E-308);
- NaN - значение, не являющееся числом;
- NEGATIVE_INFINITY - минус бесконечность;
- POSITIVE_INFINITY - плюс бесконечность.

Свойство toString([основание]) преобразует число в десятичную систему счисления (или в число в системе по основанию «основание») и возвращает в виде строки. Свойство valueOf() возвращает двоичное численное значение числа.

7.3.1.8 Объект «String» дает вам набор методов для работы с текстом. Для создания объекта используется такой синтаксис:

MyString = new String([значение]);

где значение - необязательный параметр: текст, представляющий собой начальное значение строковой переменной. Если это число, оно сначала преобразуется в строковый формат. Объект имеет единственное свойство length указывающее количество символов в строке.

Для работы с объектом используются следующие методы:

Таблица 7.7 – Методы для работы со строками

Метод	Описание
charAt(позиция)	Возвращает символ, стоящий в позиции *позиция* строки.
indexOf(значение [, позиция])	Возвращает позицию первого появления строки *значение*, начиная с позиции *позиция.*
lastIndexOf(значение [, позиция])	Возвращает позицию первого с конца строки появления строки *значение*, начиная с позиции *позиция.*
split(разделитель)	Возвращаем массив строк, созданный разделением всех встретившихся строк разделительным символом *разделитель.*
substring(номе1, номер2)	Возвращает подстроку данного объекта **String** , начинающуюся символом в позиции *номер1* и заканчивающуюся символом в позиции **перед** *номер2.*

toLowerCase()	Возвращает исходную строку со всеми символами, преобразованными в строчные.
toUpperCase()	Возвращает исходную строку со всеми символами, преобразованными в заглавные.

7.4 Ввод-вывод, управление программой

В JavaScript есть три разных метода для ввода информации пользователем или вывода ее на экран.

7.4.1 Метод Alert позволяет выводить окно сообщения с кнопкой «ОК». Следующий вызов метода выведет на экран сообщение, приведенное на рисунке.

alert("Hello World!");

Рисунок 7.1 – Вид окна сообщения

Возможность продолжения работы браузера только после нажатия кнопки, приводящее к закрытию окна.

7.4.2 Метод Confirm выводит окно сообщения с кнопками «ОК» и «Cancel». Возвращает True, если нажата кнопка «ОК», и False - если нажата кнопка «Cancel».

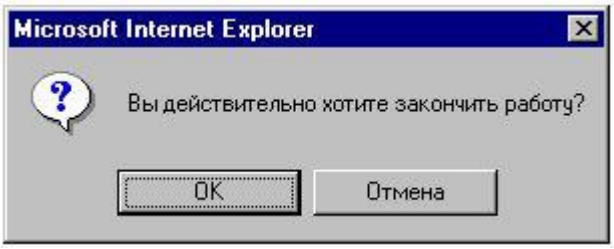

Рисунок 7.2– Вид окна сообщения с альтернативным выбором

Параметр метода – строковая переменная, выводимая в окне сообщения, например, изображенная на рисунке:

confirm("Вы действительно хотите закончить работу?");

7.4.3 Метод Prompt выводит окно сообщения и текстовое поле для ввода информации пользователем.

Первый аргумент в формате строки определяет текст, который будет находится над текстовым полем. Второй аргумент - это строка, целое число или свойство существующего объекта, формирующее значение, по умолчанию отображаемое в текстовом поле. Если второй аргумент не задан, внутри текстового поля появится надпись «undefined».

Если нажата кнопка «ОК», возвращается впечатанный в поле текст, иначе - возвращается False.

Рисунок 7.3 – Вид окна сообщения с полем ввода

Данное окно выводится при помощи метода:

prompt("Введите ваше имя", " ");

7.4.4 Для управления ходом программы в JavaScript есть два способа. Первый включает в себя условные выражения, которые в зависимости от выполненного условия выполняют либо одну часть программы, либо другую. Вторым способом является использование циклов, в которых повторяется набор команд.

В JavaScript есть лишь одна конструкция условного выражения:

if ... [then]... else... .

Она используется для выполнения различных блоков кода в зависимости от того, какое условие выполняется. Синтаксис этой конструкции таков:

```
if (условие)
  {
    код выполняется, если условие выполнено
  }
else
  {
    код выполняется, если условие не выполнено
  };
```

Основные свойства написания оператора:

- часть, определяемая словом else, необязательна;
- конструкция допускает вложенные условия;

- если одновременно указываются несколько условий, они должны быть заключены в скобки.

Ниже приведен пример использования конструкции условного выражения:

```
person_type = prompt("Кто вы?", "");
if (person_type == "кошка")
  alert("У нас есть немного кошачьей еды");
else
 {
   if (person_type == "собака")
     alert("У нас есть небольшая косточка");
 else
   {
     If (person_type == "человек")
        alert("Так заходи, поешь по-человечески!");
   };
```

Заметим, что фигурные скобки нужны лишь в случае, когда блок содержит больше одного выражения. Так же, как и в большинстве других конструкций, они могут быть опущены, если используется лишь одно выражение. Точка с запятой в конце необходима по-прежнему.

7.4.5 Простой цикл выполняется указанное количество раз, определяется оператором:

for - выполняет блок кода указанное количество раз.

Например, следующий цикл выведет последовательность десяти целых чисел:

```
for (i = 1; i = 10; i ++)
  {
   document.write(i);
  }
```

7.4.6 Условный цикл определяется оператором while. Синтаксис конструкции цикла:

```
while (условие)
{
выполняемые команды ...
}
```

Цикл будет выполняться пока условие истинно. Для принудительного выхода из цикла может использоваться оператор break.

```
if (n == "q")
  { alert("Увидимся позже!");
    break;  }
```

7.4.7 Существует возможность организации перехода на оператор, следующий за условным оператором. Такой переход осуществляется при помощи оператора continue.

В следующем примере осуществлен пропуск оператора, вставленного внутри условного выражения:

```
x = 0;
while (x !=1)
  {
    if (!(confirm("Прибавить к n единицу?")))
      { continue
        // следующая команда никогда не будет выполнена
        x ++;
      }
    x ++;
  }
alert("До свидания!");
```

7.4.8 В JavaScript существует набор встроенных функций, доступных при выполнении кода. Описание основных функций приведено в таблице:

Таблица 7.8 – Встроенные функции

Функция	**Описание**
escape(символ)	Возвращает строку в виде %XX, где XX - код ASCII символа в скобках
eval(выражение)	Возвращает результат вычисления выражения в скобках
isNaN(значение)	Возвращает True, если значение в скобках не является числом
parseFloat(строка)	Преобразует строку в скобках в число с плавающей запятой
parseInt(строка, основание)	Преобразует строку в скобках в целое число в системе счисления с указанным основанием
typeOf(объект)	Возвращает тип указанного объекта как строку, например, "boolean", "function" и т.д.

7.4.9 Любой язык программирования содержит набор служебных слов которые нельзя использовать для названия функции, метода, переменной или объекта.

Слова, которые не используются сейчас как ключевые слова JavaScript, зарезервированы на будущее. Список зарезервированных слов приведен в следующей таблице:

Таблица 7.9 – Список служебных слов

abstract	**else**	**int**	**super**

boolean	**exstends**	**interface**	**switch**
break	**false**	**long**	**synchronized**
byte	**final**	**native**	**this**
case	**finally**	**new**	**throw**
cath	**float**	**null**	**throws**
char	**for**	**package**	**transient**
class	**function**	**private**	**true**
const	**goto**	**protected**	**try**
continue	**if**	**public**	**typeof**
default	**implements**	**reset**	**var**
delete	**import**	**return**	**void**
do	**in**	**short**	**while**
double	**instanceof**	**static**	**with**

7.5 События JavaScript

Для запуска скрипта на исполнение в программе браузера должно наступить определенное событие. Все события можно разделить на несколько крупных классов:

- связанные с документом, открытие или закрытие;
- события от нажатий кнопок мыши;
- связанные с перемещением фокуса ввода;
- системные события, например, от таймера.

Всего насчитывается около ста событий. В разделе рассматриваются наиболее часто используемые события, необходимые для выполнения лабораторной работы.

В связи с развитием языка необходимо учитывать, что в области внутренних событий (например, в привязке скриптов к событиям) весьма вероятны изменения. Работа в этой области ведется членами рабочей группы по объектной модели документов W3C (W3C Document Object Model Working Group). Более подробную информацию можно найти на Web-сайте W3C по адресу http://www.w3.org/.

7.5.1 События документа:

1. Событие «load» происходит, когда браузер завершает загрузку окна или всех фреймов внутри элемента <FRAMESET>. Соответствующий обработчик события onLoad выполняет программу JavaScript.

Обработчик событий onLoad используется внутри элементов <BODY> или <FRAMESET>, например:

<BODY onLoad="имя скрипта">.

Пример загрузки часов при помощи такого обработчика приведен в пункте 7.1.4 методического пособия.

При использовании в элементе <FRAMESET> событие наступает при окончании загрузки документов во все фреймы.

2. Событие «unload» происходит, когда браузер удаляет документ из окна или фрейма. Этот атрибут может использоваться в элементах BODY и FRAMESET аналогично событию «load». Обработчик onUnload запустит скрипт в момент поступления команды на смену отображаемого документа.

<BODY onUnload="...">.

Событие onUnload, размещенное внутри фрейма в элементе <BODY>, происходит перед событием onUnload, размещенного внутри фреймосодержащей страницы в теге <FRAMESET>.

7.5.2 События от «мыши»:

1. Одно из воздействий производимых мышью – одинарное или двойное нажатие одной из кнопок. Как правило, рассматривается нажатия левой кнопки.

Второй тип воздействия – это перемещение курсора. Перемещаясь по документу, курсор проходит рабочее поле, выделенное браузером под конкретный элемент, что рассматривается системой как событие.

2. Основное событие «click» происходит при однократном щелчке мышью на элементе HTML – документа. Обрабатывают это событие все отображаемые браузером (видимые) элементы. Обработчик этого событий onClick выполняет программу JavaScript. Особенно часто данное событие применяется к элементам электронных форм - LABEL, INPUT, SELECT, TEXTAREA и BUTTON, а так же элементу ссылок.

3. Событие «dblclick» соответствует двойному щелчку мыши на выбранном элементе. Если в обычных программных средствах этот элемент событий используется довольно часто, при разработке Web – приложений, стараются его не применять. Соответствующий обработчик – onDblclick. Как и предыдущий обработчик, onDblclick можно использовать с большинством видимых элементов.

4. Следующие два события служат для разделения операции щелчка кнопки на два этапа – нажатие и отпускание кнопки. Событие при нажатии – «mousedown», а при отпускании – «mouseup».

Соответственно обработчики onMousedown и onMouseup используются аналогично обработчикам от щелчка кнопки.

5. Второй тип воздействия представлен тремя событиями:

- «mouseOver», происходит каждый раз, когда курсор мыши попадает на объект;
- «mouseMove», происходит при перемещении курсора мыши, когда он находится на элементе;
- «mouseOut», происходит при перемещении курсора мыши за пределы элемента.

Обработчики событий onMouseOver, onmouseMove и onmouseOut выполняют указанные скрипты, когда происходят соответствующие события.

7.5.3 События от фокуса объекта:

1. Фокус – свойство управляющих элементов. Обычно фокус отображается мигающим курсором и называется фокусом ввода, поскольку курсор указывает местоположение знака вводимого с клавиатуры.

Передвижение между несколькими управляющими элементами осуществляется или при помощи мыши, или нажатием клавиши «Tab». Первый позволяет перемещаться в любой последовательности, второй – в порядке следования управляющих элементов.

2. Событие «focus» происходит, когда поле получает фокус ввода с клавиатуры, соблюдая последовательность переходов, или щелчком мыши. Выбор результатов внутри поля связано с событием «select», но не с событием «focus».

При попадании фокуса на поля управляющих элементов INPUT, SELECT, TEXTAREA и BUTTON, обработчик событий onFocus выполняет программу JavaScript.

3. Соответственно при потере фокуса ввода элементом происходит событие «blur». Оно происходит, не зависимо было, изменено значение поля пользователем, или нет. Как и предыдущее событие наступает при каждом потере фокуса, сколько бы раз пользователь не проходил курсором данное поле. Соответствующий обработчик onBlur, запускает на исполнение указанный скрипт.

4. Последнее событие «change» происходит, когда поля формы select, text или textarea теряют фокус и их значения изменяются, с момента попадания на него фокуса. Обработчик событий onChange выполняет программу JavaScript, назначение которой, как правило, проверка введенных пользователем данных и подтверждение ввода.

7.5.4 События форм:

1. Событие «submit» происходит, когда пользователь отправляет данные формы на Web-сервер. Именно в это время обработчик событий onSubmit, используемый только с элементом FORM, выполняет указанный скрипт. Отправка данных, как правило, производиться нажатием клавиши «Отправить», определяемой элементом INPUT типа submit. Но, скрипт выполняемый по событию «click», этой клавиши, отличается от скрипта, выполняемого при отправки данных.

Для проверки корректности данных необходимо использовать именно событие «submit». Использование оператора return в коде обработчика, возвращающего false, позволяет отменить передачу данных на сервер. В то время как скрипт, исполняющийся при нажатии клавиши, не может влиять на отправку данных.

2. Аналогично, при нажатии клавиши «Очистить», определенной элементом INPUT типа reset, происходит событие «reset». Обработчик onReset, определяемый в элементе FORM, может использоваться для переопределения условий очистки полей формы.

3. У полей элементов управления INPUT и TEXTAREA есть еще один механизм событий, когда пользователь выделяет фрагмент текста – событие «select». Выделение производиться с использованием нажатия кнопки и одновременного движения курсора. Аналогичные действия можно производить и при помощи служебных клавиш клавиатуры. Обработчик события onSelect запустит скрипт при завершении действия, при отпускании кнопки мыши.

7.5.5 События клавиатуры:

1. При передвижении курсора по окну браузера, он поочередно перемещается между элементами HTML – документа, указывая в каждый момент времени на определенный элемент. Разумеется, речь идет об отображаемых элементах. И хотя чаще всего, события клавиатуры, используются с управляющими элементами, поддерживаются практически всеми элементами языка - A, ACRONYM, ADDRESS, APPLET, AREA, B, BIG, BLOCKQUOTE, BODY, BUTTON, CAPTION, CENTER, CITE, CODE, DD, DEL, DIR, DIV, DT, EM, FONT, FORM, H1 - H6, HR, I, IMG, INPUT, LABEL, LI, MAP, MARQUEE, OBJECT, OL, P, PRE, Q, S, SAMP, SELECT, SMALL, SPAN, STRIKE, STRONG, SUB, SUP, TABLE, TBODY, TD, TEXTAREA, TFOOT, TH, THEAD, TR, TT, U, UL, VAR, document.

2. Первое событие «keypress» наступает при нажатии и отпускании клавиши в то время, когда курсор находится в поле отображения элемента. Событие onKeypress при отпускании клавиши запустит скрипт.

3. Второе и третье событие разделяют предшествующее на два этапа. Нажатие клавиши – «keydown» и отпускание - «keyup». Соответственно обработчик onKeydown запустит скрипт при нажатии клавиши, а обработчик onKeyup при отпускании. Скрипт может быть один и тот же, в этом случае он выполниться дважды.

Примеры вызова различных событий приведены в данной работе и в коде скриптов приложения.

7.6 Пример лабораторной работы

Пример выполнения лабораторной работы использует элемент TEXTAREA - Многострочное текстовое поле ввода. Для реализации выбраны атрибуты:

cols - определяет ширину поля в символах;
disabled - делает элемент недоступным;
readonly - текст только для чтения. Нельзя изменять, вставлять и т.д.;
rows - определяет число строк;
title - всплывающая подсказка.

Для управления стилевым оформлением задается цвет и размер шрифта вводимого текста.

7.7.1 Страница ввода значений атрибутов и свойств стилей должна содержать краткое описание назначения параметров и их граничные значения.

Форма для ввода параметров приблизительно содержит следующий код:

```
<html>
<head>
<title>Лабораторная по скриптам. TEXTAREA</title>
<meta http-equiv="Content-Type" content="text/html; charset=Windows-1251">
</head>
<body bgcolor="#FDF0FB" text="#000000">
<script language="JavaScript">
<!--//////////////////////////
function gen(c,r,txt,rd,ttl,dis,sz,cl)
{
if(ttl=="")                alert("Введите значение атрибута title");
else {
if(txt=="")                alert("Введите текст в поле TEXTAREA");
else { var p=isNaN(c);
```

```
if((p==true)||(c==""))      alert("Введите числовое значение в поле 'cols'");
else {
var p=isNaN(r); if((p==true)||(r==""))
                                alert("Введите числовое значение в поле 'rows'");
else {
var ttab = window.open("","gentb","height=300,width=400").document
ttab.write("<html>\n<head>\n<title>TEXTAREA</title>\n<body>
<TEXTAREA cols=" +c+ " rows=" +r+" style='font-size: " +sz+";
color: "+cl+"' title=' " + ttl+" ' ")
if (rd!="нет") ttab.write("  readonly ")
if (dis!="нет") ttab.write("  disabled ")
ttab.write(">  "+txt+"
</TEXTAREA>\n</body>\n</html>")  }}}}}//------></script><br>
```
<table width="730" border="2" align="center">
<tr><td width="100%" bgcolor="#FAF768" align="center">
<H2>Проектирование скиптов (JavaScript)</H2></td></tr></table>
<H3>Многострочное текстовое поле ввода - TEXTAREA.</H3>

Элемент TEXTAREA создает многострочное поле для ввода текста. По умолчанию создается пустое поле шириной в 20 символов и состоящее из двух строк. Выводимый текст находится между тегами. Требуется закрывающий тег.

<h4>Атрибуты </h4>
cols - Определяет ширину поля в символах
disabled - Делает элемент недоступным
readonly - Текст только для чтения. Нельзя изменять, вставлять и т.д.
rows - Определяет число строк
title - Всплывающая подсказка
Общий вид кода многострочного текстового поля :
<P style="color:red; margin-left:100"><TEXTAREA NAME="n1" cols=11 rows=7>
Содержание поля ввода по умолчанию
</TEXTAREA></P>

<DIV align="center">
Пример. Пользователь имеет возможность изменять стиль вводимого текста
<form name="gt6">
<LABEL> cols= <INPUT name="n20" type="text"> </LABEL>
<LABEL> rows= <INPUT name="n21" type="text"> </LABEL>
<LABEL>Установить атрибут readonly <select name="read9">
<option value="нет" selected>нет</option>
<option value="да" >да</option></select> </LABEL>
<LABEL>Установить атрибут disabled <select name="d4">
<option value="нет" selected>нет</option>
<option value="да" >да</option></select></LABEL>
<LABEL> Значение атрибута title <INPUT name="n30" type="text"> </LABEL>
<LABEL> Цвет <select name="clr">
<option value="#00FFFF">aqua</option>
<option value="#000000" selected>black</option>
<option value="#A52A2A">brown</option>
<option value="#FF7F50">coral</option>
<option value="#FFD700">gold</option>
<option value="#008000">green</option>
<option value="#808000">olive</option>
```
```

```
        <option value="#FFA500">orange</option>
        <option value="#FF0000">red</option></select></LABEL>   
<LABEL> Размер шрифта <select name="size">
        <option value="8pt">8</option>
        <option value="14pt" selected>14</option>
        <option value="16pt">16</option>
        <option value="20pt">20</option></select></LABEL><br><br>
<TEXTAREA name="n31" cols=30 rows=7></TEXTAREA><br><br>
<input type="button" value="Генерировать многострочное текстовое поле"
onClick="gen(document.gt7.n20.value,document.gt7.n21.value,document.gt7.n31.value,docume
nt.gt7.read9.value,
document.gt7.n30.value,
document.gt7.d4.value,document.gt7.size.value,document.gt7.clr.value)">
</DIV> </form> </body> </html>
```

7.7.2Для ввода значений некоторых параметров используется элемент формирования меню выбора. Числовые параметры вводятся в поля ввода текста. Отображение данной формы в окне браузера приведено на рисунке:

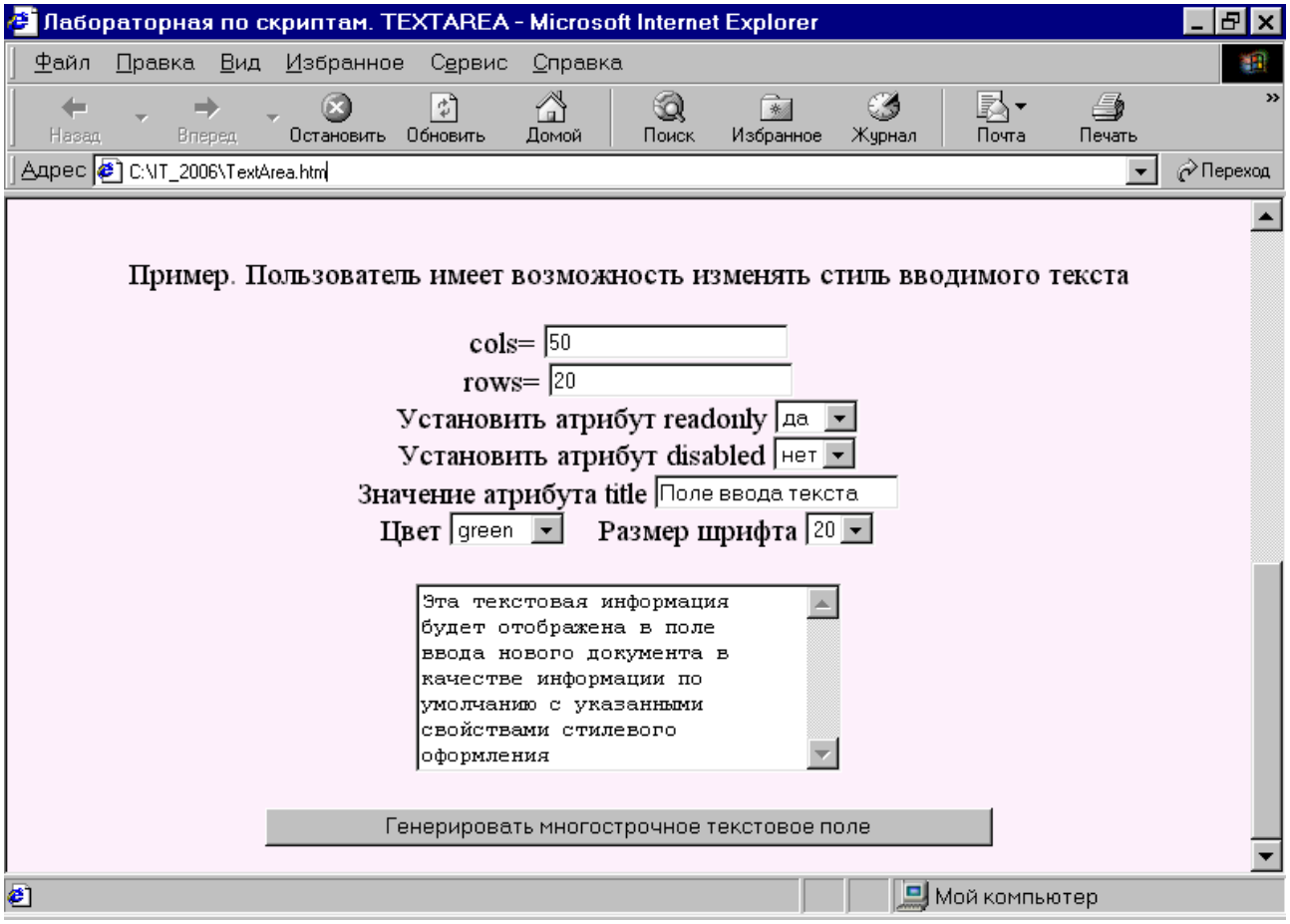

Рисунок 7.4 – Форма ввода параметров элемента

Для генерации объекта используется событие нажатия клавиши:

onClick="gen(…)"

В обработчике, которого вызывается скрипт генерации нового окна браузера с определенным кодом.

7.7.3 Скрипт, генерирующий код страницы, может содержать любое необходимое количество вложенных циклов для демонстрации набора элементов.

В теле скрипта генерируется новый объект как переменная, текст которой в последствии заполняется кодом страницы.

var ttab = window.open("","gentb","height=600,width=800").document

Обратить внимание на параметры нового окна и его возможность вместить генерируемое изображение элемента.

7.7.4 Результат работы скрипта отображается, как правило, в новом окне браузера. Допускается генерация элемента в одном из фреймов основного окна. В отчете приводится рисунок с новым объектом и код отображаемого документа:

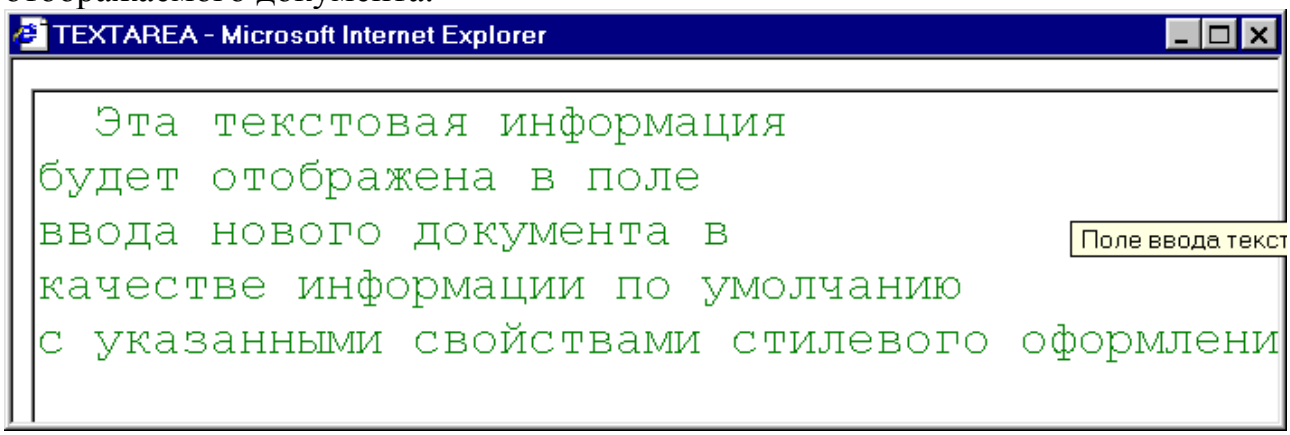

Рисунок 7.5 – Форма ввода параметров элемента

Скрипт, приведенный в примере, основываясь на введенных в поля формы данных, генерирует следующий код документа:

```
<html><head>
<title>TEXTAREA</title>
</head><body>
<TEXTAREA cols=50 rows=20 style='font-size: 20pt; color: #008000'
title=' Поле ввода текста '   readonly >  Эта текстовая информация
будет отображена в поле ввода нового документа в качестве информации
по умолчанию с указанными свойствами стилевого оформления </TEXTAREA>
</body></html>
```

7.7.4 Генерация нового объекта Window. Самый верхний объект в иерархии с двумя основными методами open(), close() и несколькими второстепенными:

- focus() – передача окну фокуса, в основном для выполнения обработчика onFocus, если такой определен в теле документа;
- setTimeout() – периодически повторяет указанную функцию;

- blur() – делает окно неактивным.

Рассмотрим открытие нового окна - функцию window.open.

```
window.open("","gentb","height=600,width=800").document
```

Первый параметр (в нашем случае, это пустая строка - "") - URL , то есть адрес документа, который должен быть помещен в новое окно. Поскольку URL не определен будет открыто пустое окно, в свойстве document которого будет записываться HTML – код.

Второй параметр (в нашем случае это "gentb") - имя нового окна. Используя имя в качестве значения атрибута target элементов <A> или <FORM>, можно вывести документы в это окно.

Третий параметр состоит из следующих возможных числовых или логических значений:

- width = NNN - ширина окна;
- height = NNN - высота окна;
- toolbar = {no | yes} - панель инструментов браузера;
- location = {no | yes} - поле для ввода URL в браузер;
- directories = {no | yes} - кнопки каталогов;
- status = {no | yes} - строка состояния;
- menubar = {no | yes} - линейка меню браузера;
- scrollbars = {no | yes} - полоса прокрутки.

В строке, содержащей третий параметр, не должно быть пробелов, все параметры и значения записываются в кавычках слитно. Пример открытия нового окна по ссылке:

```
<A href="javascript:" onMouseOver="win1 =
window.open('test.html','w1','width=90,height=80,toolbar=yes,location=yes,
directories=yes,status=yes,menubar=yes,scrollbars=yes,'); return true;"
onMouseOut=" win1.window.close(); return true;">
<font>A теперь подведите курсор сюда!</font></A>
```

При наведении курсора на текст гиперссылки будет открываться новое окно браузера с документом test,html, с указанными параметрами.

7.7.5 При запуске кода скрипта, браузер может обнаружить ошибку. В этом случае генерируется специальное сообщение с указанием строки и столбца символа, вызвавшего ошибку.

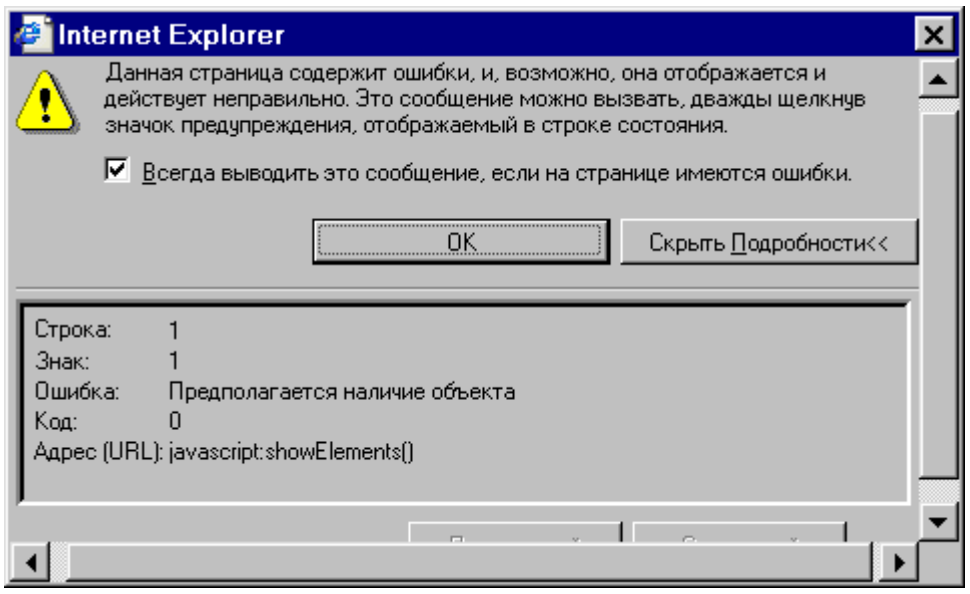

Рисунок 7.6 – Окно сообщения об ошибке

Наличие такого сообщения, показывает, что алгоритм скрипта не будет выполнен. Следовательно, все действия, предусмотренные в коде, будут отменены.

7.7.6 Проверка значений элементов формы. Скрипт, приведенный в примере содержит один недостаток. Поля формы позволяют вводить любые символы. Например, если в поле ввода толщины границы (border) ввести символы отличные от цифр, то граница генерироваться не будет. Так же не контролируется обязательное заполнение значений элементов.

В задачу, реализуемую в рамках лабораторной работы, входит обязательная реализация нескольких условий контроля значений атрибутов. При нарушении условий заполнений формы, пользователю выводится одно из служебных сообщений, смотри раздел 7.4. Обработку введенных значений можно проводить при заполнении конкретного поля или при отправке на сервер в обработчике onSubmit элемента FORM. В последнем случае скрипт должен заканчиваться вызовом метода:

document.form.submit;

7.7.7 Проверка корректности введенных данных может проводиться по следующим условиям:

1. Проверка ввода значения (не пустое поле) при помощи оператора:

```
if (document.form.userid.value != "");
```

где userid – имя элемента ввода.

2. Проверка вхождения в заданный диапазон вводимого пользователем значения, например, возраст человека:

```
u1 = (documrnt.form.age.value > 18) & (documrnt.form.age.value < 60);
```

где: u1 – логическая переменная, age – имя поля ввода.

3. Проверка наличия определенного знака во вводимом значении строки текста, например наличие знаков «@» и «**.**» в адресе электронной почты:

```
em = document.form.email.value;
u2 = ((em.indexOf("@") != -1) & (em.indexOf(".") != -1);
```

где: em – строковая переменная, u2 – логическая переменная, email – имя поля ввода адреса.

4. Контроль ввода числового значения осуществляется вызовом специального метода:

```
if (isNaN(document.form.varint.value));
```

где: isNaN – функция проверки соответствия значения числу, varint – имя поля.

При наличии нескольких условий проверки, строится логическое произведение условий:

```
if ((u1) & (u2) & (u3)) document.form.submit;
else alert ("сообщение о ошибке ввода");
```

Если проверка осуществляется непосредственно при заполнении поля ввода используются обработчики событий от нажатия клавиш клавиатуры или события, связанные с потерей фокуса ввода элементом.

7.7.8 Вопросы для защиты:

1. Форматы определения скриптов в HTML – документах.
2. Основные объекты языка написания скриптов и типы переменных.
3. Иерархия объектов, используемых в языке.
4. Определение функции и метода.
5. Методы вывода информации пользователю.
6. Операторы управления ходом выполнения программы.
7. Последовательность определения нового объекта window.
8. Обращение к свойствам элементов документа.
9. Получение значений управляющих элементов форм.
10. Обращение к свойствам стилевого оформления элементов.

ПРИЛОЖЕНИЕ А

Примеры скриптов

В приложении приведены примеры сложных скриптов, реализующих алгоритмы с применением распространенных функций и методов JavaScript. Изучение кода поможет разобраться с методикой использования сложных методов. Все примеры могут быть воспроизведены при реализации собственных страниц задания лабораторной работы.

А.1 Пример простого калькулятора. Скрипт выполняет функцию сложения двух чисел, вводимых пользователем в поля ввода. При нажатии клавиши в третьем поле отображается результат сложения. Для перевода чисел в десятичную форму используется стандартная функция parseInt().

```
        <script  language="JavaScript">
function myFunction4() {
   var val1 = parseInt(document.myForm4.myNum1.value);
   var val2 = parseInt(document.myForm4.myNum2.value);
   var val3 = val1 + val2;
   document.myForm4.myNum3.value = val3; }
</script>
<form name="myForm4">
  Сложение 2 чисел:<p>
  Введите первое слагаемое:
  <input type="num" name="myNum1" value=0 size=10><p>
  Введите второе слагаемое:
  <input type="num" name="myNum2" value=0 size=10><p>
  <input type="button" name="Button4"
    value="Сложение" onclick="myFunction4(); return true;"><p>
   <input type="num" name="myNum3" value=0 size=10><p>
</form>
```

А.2 Данный пример демонстрирует обращение к свойствам браузера, посредством встроенного объекта языка - Navigator.

```
<html><head>
<title>Браузер</title>
</head>
<body bgcolor="ffffff">
<center><table border=1>
<tr><td>Имя броузера</td><td>
<script language="JavaScript">
document.write(navigator.appName);</script></td>
<tr><td>Версия броузера</td><td>
```

```
<script language="JavaScript">
document.write(navigator.appVersion);</script></td>
<tr><td>Кодовое название броузера</td><td>
<script language="JavaScript">
document.write(navigator.appCodeName);</script></td>
<tr><td>Заголовок пользовательского <br>агента</td><td valign=top>
<script language="JavaScript">
document.write(navigator.userAgent);</script></td>
</table></center>
</body></html>
```

А.3 Набор all объекта document представляет все объекты в иерархии документа. Каждый объект в этом наборе представляется как программируемый объект. Доступ к каждому объекту осуществляется по порядковому номеру или по ID (идентификатор, уникальное имя).

```
        <SCRIPT language="JavaScript">
function showElements()
{
var tag_names = "";
for (i=0; i<document.all.length; i++)
        tag_names = tag_names + document.all(i).tagName + " ";
        alert("This document contains: " + tag_names);
        } </SCRIPT>
        <BODY onload="showElements()">
```

Данный скрипт, при загрузке документа, перебирая по порядковым номерам массив document.all, составит список всех значений свойств tagName. Это свойство определяет название элемента. Сформированная строка выводится в диалоговое окно, приведенное на рисунке:

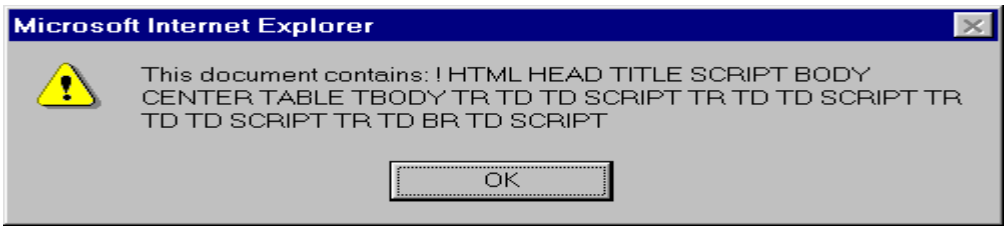

Рисунок А.1 – Форма вывода списка элементов

Набор all определяет лишь порядок следования элементов в нем, но не иерархию элементов. (Заметьте также, что в списке не приводятся закрывающие тэги). Он отображает порядок следования элементов в коде HTML.

Набор all представляет текущее состояние документа, и немедленно обновляется при изменении содержимого. То есть вы можете в реальном

времени добавлять и удалять элементы, при этом обновление набора происходит автоматически.

А.4 Следующий пример используется для построения иерархии объектов документа.

Вы всегда можете проверить, является ли тот или иной объект родителем (старшим в иерархии) другого, и наоборот, имеет ли данный элемент родителя. Соответствующие методы - contains и parentElement.

```
<SCRIPT language="JavaScript">
function showHierarchy() {
var depth = 0;
var msg = document.all(0).tagName;
for (i=1; i<document.all.length; i++) {
if (document.all(i-1).contains(document.all(i))==true)
{ depth = depth + 1; } else {
var elParent = document.all(i-1).parentElement;
for ( ; depth>0; depth--) {
if (elParent.contains(document.all(i))==true)
break;
elParent = elParent.parentElement; }  }
msg = msg + "\n";
for (j=1; j<=depth; j++)
msg = msg + " ";
msg = msg + document.all(i).tagName; }
alert("Этот документ содержит:\n" + msg); }
</SCRIPT>
```

Результатом работы скрипта является окно сообщения с изображением иерархического дерева элементов:

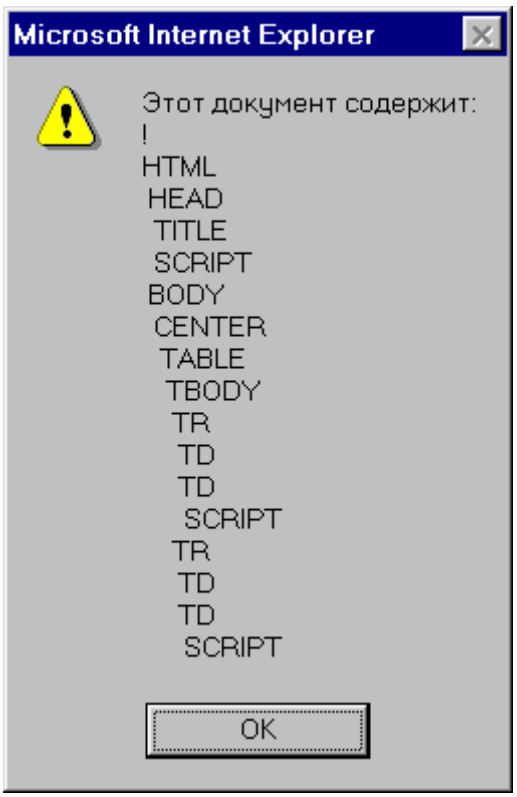

Рисунок А.2 – Форма вывода иерархии элементов

Как и в предыдущем примере, не отображаются закрывающие теги. Верхним элементом всегда является HTML.