Лабораторный курс по языку программирования JavaScript

Содержание

1 Основные положения
2 Функция и обработка событий
3 Организация ветвлений в программах
4 Методы в JavaScript
5 Переключатели
6 Флажки
7 Списки
8 Фреймы
9 Повторяющиеся вычисления - циклы
10 Обработка и представление дат
11 Работа со строками
12 Массивы и методы работы с ними
13 Формы
14 Объект Image

Лабораторная работа №1
Основные положения

Программа (сценарий) на языке JavaScript представляет собой последовательность операторов с "точкой с запятой" (;) между ними. Если каждый оператор размещается на одной строке, то разделитель можно не писать. Один оператор может располагаться на нескольких строках.

В программах на JavaScript можно использовать комментарии. Для того чтобы задать комментарий, располагающийся на одной строке, достаточно перед его текстом поставить две косые черты (//). Если же поясняющий текст занимает несколько строк, то его следует заключать между символами /* и */. В JavaScript строчные и прописные буквы алфавита считаются разными символами. Любой язык программирования оперирует с постоянными и переменными величинами. В JavaScript это литералы и переменные.

Опр.: Простейшие данные, с которыми может оперировать программа, называются литералами.

Литералы не могут изменяться. Литералы целого типа могут быть заданы в десятичном (по основанию 10), шестнадцатеричном (по основанию 16) или восьмеричном (по основанию 8) представлении. Шестнадцатеричные числа включают цифры 0-9 и буквы a, b, c, d, e, f. Шестнадцатеричные числа записываются с символами 0x перед числом, например, 0x25, 0xa1,0xff. Запись вещественного литерала отличается от записи вещественного числа в математике тем, что вместо запятой, отделяющей целую часть от дробной, указывается точка, например, 123.34, -22.56. Кроме того, для записи вещественных чисел можно использовать так называемую экспоненциальную форму.

Кроме целых и вещественных значений в языке JavaScript могут встречаться так называемые логические значения. Существуют только два логических значения: истина и ложь. Первое представляется литералом true, второе - false. В некоторых реализациях JavaScript может быть использована единица в качестве true, и ноль в качестве false.

Строковый литерал представляется последовательностью символов, заключенной в одинарные или двойные кавычки. Примером строкового литерала может быть строка "результат" или 'результат'.

Опр.: Элемент, используемый для хранения данных, называется переменной.

Тип переменной зависит от хранимых в ней данных, при изменении типа данных меняется тип переменной. Определить переменную можно с помощью оператора var, например: var test1.

В данном случае тип переменной test1 не определен и станет известен только после присвоения переменной некоторого значения. Оператор var можно использовать и для инициализации переменной, например, конструкцией var test2=276 определяется переменная test2 и ей присваивается значение 276.

Значение переменной изменяется в результате выполнения оператора присваивания. Оператор присваивания может быть использован в любом месте программы и способен изменить не только значение, но и тип переменной. Оператор присваивания выглядит так: a=b, где a - переменная, которой мы хотим задать некоторое значение; b - выражение, определяющее новое значение переменной.

Пусть в сценарии описаны следующие переменные

```
var n=3725
 var x=2.75
 var p=true
 var s="Выполнение завершено"
```

n и x имеют тип number, тип переменной p - логический, переменная s имеет тип string. В JavaScript определен тип function для всех стандартных функций и функций, определяемых пользователем. Объекты JavaScript имеют тип данных object. Переменные типа object часто называют просто объектами, они могут хранить объекты.

Опр.: Выражения строятся из литералов, переменных, знаков операций, скобок. В зависимости от типа вычисленного значения выражения можно разделить на арифметические, логические и строковые. Арифметические выражения получаются при выполнении операций, перечисленных в табл.1.

Операции отношения применимы к операндам любого типа. Результат операции - логическое значение true, если сравнение верно, и false - в противном случае.

Приоритет операций определяет порядок, в котором выполняются операции в выражении. В табл.2 перечислены рассмотренные операции в порядке убывания приоритетов.

Сценарии, написанные на языке JavaScript, могут располагаться непосредственно в HTML-документе между тегами <script> и </script>.

Одним из параметров тега <script> является language, который определяет используемый язык сценариев. Для языка JavaScript значение параметра равно "JavaScript". Если применяется язык сценариев VBScript, то значение параметра должно быть равным "VBScript". В случае использования языка JavaScript параметр language можно опускать, т. к. этот язык выбирается браузером по умолчанию.

Обычно браузеры, не поддерживающие какие-либо теги HTML, эти теги просто игнорируют. Попытка браузера проанализировать содержимое не поддерживаемых тегов может привести к неверному отображению страницы. Чтобы избежать такой ситуации, рекомендуется помещать операторы языка JavaScript в теги комментария <!-- ... -->. Для правильной работы интерпретатора перед закрывающим тегом комментария --> следует поставить символы //.

Итак, для размещения сценария в HTML-документе следует написать следующее:

<script language="JavaScript">

</script>

Документ может содержать несколько тегов <script>. Все они последовательно обрабатываются интерпретатором JavaScript. В следующем примере в раздел <body> (в тело) HTML-документа вставлены операторы языка JavaScript.

Пример 1. Вычисление площади треугольника

Необходимо написать сценарий, определяющий площадь прямоугольного треугольника по заданным катетам. Сценарий разместим в разделе <body> HTML-документа (листинг 1).

Листинг 1. Первый сценарий в документе :

```
<HTML>
<HEAD>
<title>Первый сценарий в документе</title>
</HEAD>
<BODY>
<P>Страница, содержащая сценарий.</P>
<script>
<!--
var a=8; h=10 /*Инициализируются две переменные*/
document.write ("Площадь прямоугольного треугольника равна ", a*h/2,".")
/*Для формирования вывода используется метод write объекта document*/
//-->
</script>
<P>Конец формирования страницы, содержащей сценарий</P>
</BODY>
</HTML>
```

Задания

1. Проверить пример из лабораторной работы.
2. Составить сценарий, в котором вычисляется площадь круга по заданному радиусу.
3. Составить сценарий, вычисляющий гипотенузу по заданным катетам.

Лабораторная работа №2
Функция и обработка события

Основным элементом языка JavaScript является функция. Описание функции имеет вид

function F (V) {S},

где F - идентификатор функции, задающий имя, по которому можно обращаться к функции; V - список параметров функции, разделяемых запятыми; S - тело функции, в нем задаются действия, которые нужно выполнить, чтобы получить результат. Необязательный оператор return определяет возвращаемое функцией значение. Обычно все определения и функции задаются в разделе <head> документа. Это обеспечивает интерпретацию и сохранение в памяти всех функций при загрузке документа в браузер.

Пример 1. Нахождение площади треугольника.

В предыдущих примерах пользователю не предоставлялась возможность вводить значения, и в зависимости от них получать результат. Интерактивные документы можно создавать, используя формы. Предположим, что мы хотим создать форму, в которой поля Основание и Высота служат для ввода соответствующих значений. Кроме того, в форме создадим кнопку Вычислить. При щелчке мышью по этой кнопке мы хотим получить значение площади треугольника. Действие пользователя (например, щелчок кнопкой мыши) вызывает событие. События в основном связаны с действиями, производимыми пользователем с элементами форм HTML. Обычно перехват и обработка события задается в параметрах элементов форм. Имя параметра обработки события начинается с приставки on, за которой следует имя самого события. Например, параметр обработки события click будет выглядеть как onclick.

Листинг 1. Реакция на событие Click.

```
<HTML>
<HEAD>
<title>Обработка значений из формы</title>
<script language="JavaScript">
<!--//
function care (a, h)
{
var s=(a*h)/2;
document.write ("Площадь прямоугольного треугольника равна ",s);
return s
}
//-->
```

```
</script>
</HEAD>
<BODY>
<P>Пример сценария со значениями из формы</P>
<FORM name="form1">
Основание: <input type="text" size=5 name="st1"><hr>
Высота: <input type="text" size=5 name="st2"><hr>
<input type="button" value=Вычислить
onClick="care(document.form1.st1.value, document.form1.st2.value)"> /*По клику мыши на кнопке в функцию care передаются два параметра - содержимое полей ввода*/
</FORM>
</BODY>
</HTML>
```

При интерпретации HTML-страницы браузером создаются объекты JavaScript. Взаимосвязь объектов между собой представляет иерархическую структуру. На самом верхнем уровне иерархии находится объект windows, представляющий окно браузера. Объект windows является предком или родителем" всех остальных объектов. Каждая страница кроме объекта windows имеет объект document. Свойства объекта document определяются содержимым самого документа: цвет фона, цвет шрифта и т. д. Для получения значения основания треугольника, введенного в первом поле формы, должна быть выполнена конструкция

```
document.forml.stl.value
```

т.е., говоря русским языком (при этом читаем с конца), используем данные value из поля ввода с именем st1 находящегося на форме form1 объекта document.

Пример 2. Вычисление площади квадрата.

Напишем сценарий, определяющий площадь квадрата по заданной стороне. Площадь должна вычисляться в тот момент, когда изменилось значение его стороны. Пусть форма содержит два текстовых поля: одно для длины стороны квадрата, другое для вычисленной площади. Кнопка Обновить очищает поля формы. Площадь квадрата вычисляется при возникновении события change, которое происходит в тот момент, когда значение элемента формы с именем num1 изменилось, и элемент потерял фокус. HTML-код представлен в примере 2.

Листинг 2. Реакция на событие Change

```
<HTML>
```

```
<HEAD>
<title>Обработка события Change - изменение значения элемента</title>
<script>
function srec(obj)
{obj.res.value=obj.num1.value* obj.num1.value}
</script>
</HEAD>
<BODY>
<P>Вычисление площади квадрата</P>
<FORM name="form1">
Сторона: <input type="text" size=7 name="num1"
onChange="srec(form1)">
<hr>
Площадь: <input type="text" size=7 name="res"><hr>
<input type="reset" value=Обновить>
</FORM>
</BODY>
</HTML>
```

Событие Focus возникает в момент, когда пользователь переходит к элементу формы с помощью клавиши <Tab> или щелчка мыши. Событие "потеря фокуса" (Blur) происходит в тот момент, когда элемент формы теряет фокус. Событие select вызывается выбором части или всего текста в текстовом поле. Например, щелкнув дважды мышью по полю, мы выделим поле, наступит событие select, обработка которого приведет к вычислению требуемого значения. В табл.3 представлены события и элементы документов HTML, в которых эти события могут происходить. В языке JavaScript определены некоторые стандартные объекты и функции, пользоваться которыми можно без предварительного описания. Одним из стандартных объектов является объект Math. В свойствах упомянутого объекта хранятся основные математические константы, а его методы можно использовать для вызова основных математических функций. В табл.4 приведены некоторые методы объекта Math. Выражение y=log x запишется y=Math.log(x).

Задания

1. Проверить примеры из лабораторной работы.
2. На плоскости заданы координаты трех точек. Напишите сценарий, который вычисляет площадь треугольника (использовать событие Focus).
3. Напишите сценарий, который для точки, заданной координатами на плоскости, определяет расстояние до начала координат (использовать событие Select).
4. Напишите сценарий, который обменивает местами значения двух введенных переменных (использовать событие Blur).

Лабораторная работа №3
Организация ветвлений в программах

При составлении программы часто необходимо выполнение различных действий в зависимости от результатов проверки некоторых условий. Для организации ветвлений можно воспользоваться условным оператором, который имеет вид:

if B {S1} else {S2}

где B - выражение логического типа; S1 и S2 - операторы. Выполнение условного оператора осуществляется следующим образом. Вычисляется значение выражения B. Если оно истинно, то выполняются операторы S1, если ложно - операторы S2. Если последовательность операторов S1 или S2 состоит лишь из одного оператора, то фигурные скобки можно опустить. Возможна сокращенная форма условного оператора:

if B {S}

где B - выражение логического типа; S - последовательность операторов. Выполнение краткого условного оператора осуществляется так: вычисляется значение выражения B, если оно истинно, то выполняются операторы S.

Пример 1. Нахождение максимального значения

Для трех заданных значений a, b, c необходимо написать сценарий, определяющий максимальное значение. Поступим следующим образом. Сначала максимальным значением m будем считать значение a, далее значение b сравним с максимальным. Если окажется, что b больше m, то максимальным становится b. И, наконец, значение c сравнивается с максимальным значением из предыдущих значений a и b. Если c больше m, то максимальным становится c. Оператор присваивания

obj.res.value=m

обеспечивает запись вычисленного максимального значения в соответствующее поле формы. Функция Number (s) преобразует объект s, заданный в качестве параметра, в число. Полностью сценарий может быть записан так, как представлено в листинге 1.

Листинг 1. Вычисление максимального значения из трех заданных

```
<HTML>
 <HEAD>
 <TITLE>Вычисление максимального значения</TITLE>
 <script language="JavaScript">
 <!-- //
 function maxval (obj )
 {
 var a = Number(obj.num1.value);
 var b = Number(obj.num2.value);
```

```
var c = Number(obj.num3.value);
var m=a
if (b > m) m=b
if (c > m) m=c
obj.res.value=m }
//-->
</script>
</HEAD>
<BODY>
<H4>Вычисление максимального значения</H4>
<FORM name="form1">
Число 1: <input type="text" size=8 name="num1"><hr>
Число 2: <input type="text" size=8 name="num2"><hr>
Число 3: <input type="text" size=8 name="num3"><hr>
Максимальное значение равно
<input type="button" value=Определить onClick="maxval(form1)">
<input type="text" size=8 name="res"><hr>
<input type="reset">
</FORM>
</BODY>
</HTML>
```

Решим рассмотренную задачу другим способом. Вспомним, что стандартный объект Math имеет метод max, который определяет наибольшее значение двух аргументов. Опишем функцию maxval1, которая определяет максимальное значение из трех заданных значений и использует объект Math.

```
function maxval1 (obj )
 {
 var a = Number(obj.num1.value);
 var b = Number(obj.num2.value);
 var c = Number(obj.num3.value);
 obj.res.value=Math.max(Math.max(a,b),c)
 }
```

Если бы требовалось определить максимальное из четырех заданных значений a, b, c, d, то можно было бы воспользоваться формулой

Math.max(Math.max(a,b), Math.max(c,d)).

Задания

1. Проверьте пример из лабораторной работы.
2. Вводится последовательность из пяти чисел. Напишите сценарий, в котором определяется число максимальных элементов.

3. Напишите программу, которая определяет, можно ли построить треугольник с заданными длинами сторон.

4. Точка на плоскости задается своими координатами. Определите, какой из четвертей прямоугольной системы координат принадлежит заданная точка.

Лабораторная работа №4
Методы в JavaScript

Во время интерпретации HTML-документа браузером создаются объекты JavaScript. Свойства объектов определяются параметрами тегов языка HTML. Структура документа отражается в иерархической структуре объектов, соответствующих HTML-тегам. Родителем всех объектов является объект windows, расположенный на самом верхнем уровне иерархии, он представляет окно браузера и создается при запуске браузера. Для того чтобы открыть новое окно в сценарии JavaScript и отобразить в нем новый документ, применяется метод open, для закрытия окна можно воспользоваться методом close. Метод alert объекта windows отображает диалоговое окно с текстом, переданным методу в качестве параметра. Данный метод используется в случаях проверки правильности вводимых данных с помощью формы. Свойства объекта windows относятся ко всему окну, в котором отображается документ.

Подчиненными объектами (или объектами нижнего уровня) являются объекты document, history, location, frame. Свойства объекта history представляют адреса ранее загружаемых HTML-страниц. Свойства объекта location связаны с URL-адресом отображаемого документа, объекта frame - со специальным способом представления данных.

Свойства объекта document определяются содержимым самого документа: шрифт, цвет фона, формы, изображения и т. д. Объект document в зависимости от своего содержимого может иметь объекты, являющиеся для него подчиненными или дочерними. В частности подчиненными для объекта document являются объекты form, image, link, area и др. Иерархическая структура объектов представлена на рис.1

Для каждой страницы создается один объект document, некоторые его свойства соответствуют параметрам тега <BODY>: bgColor, fgcolor, linkcolor, alinkcolor, vlinkColor. Методы write и writeln записывают в документ текст, задаваемый параметром.

Если документ содержит изображения, то доступ к объекту, определяющему изображение, можно получить с помощью переменной, указанной в параметре name тега <Img>. Объект image имеет свойство images, которое содержит ссылки на все изображения, расположенные в документе. Ссылки перенумерованы, начиная с нуля. Доступ к первому изображению можно получить с помощью составной конструкции document.images[0], ко второму - document.images[1]. Если на странице пять изображений, то доступ к последнему изображению можно получить, воспользовавшись ссылкой document.images[4].

Рассмотрим примеры, в которых используются различные свойства объектов.

Для встраивания изображений в HTML-документ служит тег <IMG>, имеющий обязательный параметр src, определяющий URL-адрес файла с изображением. Можно задавать размеры выводимого изображения. Значение

параметра width определяет ширину изображения, значение параметра height - высоту изображения. Значения параметров ширины и высоты могут не совпадать с истинными размерами изображений, тогда при загрузке изображения автоматически выполняется перемасштабирование.

Изображение можно поместить в рамку. Для этого используется параметр border. Значением параметра должно быть число, определяющее толщину рамки в пикселях. По умолчанию рамка вокруг изображения отсутствует, если только изображение не является ссылкой.

Параметр alt определяет альтернативный текст. При наведении курсора мыши на изображение появляется комментарий.

Пример 1. Перестановка изображений

Напишем сценарий, который реализует обмен рисунков в документе. Пусть в документе расположено три изображения, пронумерованных от 1 до 3. В текстовых полях указываются номера рисунков, которые необходимо поменять местами. Требуется, чтобы после нажатия кнопки Обменять изображения переместились на нужные места.

Сначала проверим, правильно ли заданы номера изображений, если это не так, то выдадим сообщение. Переменная z служит для запоминания адреса первого графического изображения. Доступ к изображению с номером r1 производится с помощью конструкции document.images[r1-1]. Для того чтобы на место изображения с номером r1 поместить изображение с номером r2, требуется выполнить оператор присваивания:

document.images[r1-1].src=document.images[r2-1].src

И, наконец, на место изображения с номером r2 помещается изображение, которое ранее было на месте с номером r1, и адрес которого запомнили в переменной z:

document.images[r2-1].src=z

Приведем полностью документ со сценарием (листинг 1).

Листинг 1. Перестановка изображений с помощью сценария

```
<HTML>
<HEAD>
<TITLE>Перестановка изображений</TITLE>
<script>
function chan(obj)
{ var r1=Number(obj.a1.value)
var r2=Number(obj.a2.value)
if ((r1<1)||(r1>3)||(r2<1)||(r2>3))
alert ("Неверно заданы номера рисунков!")
else
{ var z=document.images[r1-1].src
```

```
document.images[r1-1].src=document.images[r2-1].src;
document.images[r2-1].src=z
} }
</script>
</HEAD>
<BODY>
<CENTER>
<H4>Галерея рисунков</H4><br>
<IMG src="p1.gif" width="90" name="pic1">
<IMG src="p2.gif" width="90" name="pic2">
<IMG src="p3.gif" width="90" name="pic3"> <br>
<FORM name=form1>
Рисунки с номерами<br>
<input type="text" name="a1" size=1> и
<input type="text" name="a2" size=1><P>
<input type="button" value="Поменять местами" onClick="chan(form1)">
</FORM>
</CENTER>
</BODY>
</HTML>
```

Пример 2. Простое вертикальное меню

Напишем сценарий, реализующий вертикальное графическое меню. При наведении курсора мыши на пункт меню меняется цветовая палитра, соответствующая выделенному пункту меню.

Каждому пункту меню соответствует два изображения: первое изображение, когда пункт меню не выбран, второе - при выбранном пункте меню, цветовая палитра рисунка изменена. Графические изображения, соответствующие ситуации, когда пункты меню не выбраны, хранятся в файлах с именами pch1.gif, pch2.gif. Соответствующие им графические изображения с измененной палитрой хранятся в файлах с именами wpch1.gif, wpch2.gif.

Функция img имеет два параметра. Первый параметр задает выбор пункта меню, второй параметр - n - определяет номер пункта меню. От этого параметра зависит, какое изображение в документе требуется изменить document.images[n-1].src (вставить на этом месте рисунок "wpch"+n+".gif" или pch"+n+".gif). Имя файла формируется динамически и представляет собой конкатенацию (слияние) строк, одна из составляющих которой - значение второго параметра. Если имена файлов не подчинены общему правилу, то в функции потребуется дополнительный анализ, какой файл подгрузить. Это сделать нетрудно, зная место в документе, из которого произошел вызов функции. Документ со сценарием, реализующий вертикальное графическое меню, представлен в листинге 2.

Листинг 2. Простое вертикальное меню

```
<HTML>
<HEAD>
<TITLE>Простое вертикальное меню</TITLE>
<script language="JavaScript">
function img(n, action)
{if (action)
{document.images[n-1].src="wpch"+n+".gif"}
else
{document.images[n-1].src="pch"+n+".gif"}
}
</script>
</HEAD>
<BODY background="fon1.jpg">
<H2><FONT color="#0000ff">Содержание</FONT></H2>
<A href="tch1.htm" onmouseover="img(1,1)" onmouseout="img(1,0)">
<IMG src="pch1.gif" alt="форматирование текста" border="0" width="103" height="35"></A><br>
<A href="tch2.htm" onmouseover="img(2,1)" onmouseout="img(2,0)">
<IMG src="pch2.gif" alt="форматирование текста" border="0" width="103" height="35"></A><br>
<A href="tch3.htm" onmouseover="img(3,1)" onmouseout="img(3,0)">
<IMG src="pch3.gif" alt="форматирование текста" border="0" width="103" height="35"></A><br>
</BODY>
</HTML>
```

При попадании курсора мыши в область изображения возникает событие Mouseover, параметр обработки события onMouseOver получает значение img(p1,true).

Задания

1. Проверить примеры лабораторной работы.
2. Написать сценарий выбора из трех изображений одного, которое вставляется ниже этих трех.
3. Написать сценарий картинки с "эффектом приближения", т.е. увеличения размеров как реакция на попадание курсора мыши в поле рисунка (использовать свойства width и height).
4. Написать сценарий графического горизонтального меню с появляющейся стрелкой над пунктом, у которого находится курсор.

Лабораторная работа №5
Переключатели

Данные удобно представлять с помощью элемента управления "переключатель" (или "радиокнопка") в том случае, когда из нескольких вариантов может быть выбран лишь один.

Пример 1. Вычисление площади фигуры.

Необходимо выбрать форму фигуры и определить ее площадь.
Пусть для выбора фигуры задана следующая форма:

```
<FORM name="form1">
Введите значение
<input type="text" name="data" size=10><hr>
Укажите форму:<br>
<input type="radio" name="fv" value=1>квадрат<br>
<input type="radio" name="fv" value=2>Круг<br>
<input type="radio" name="fv" value=3>треугольник<hr>
<input type="reset" value="Отменить"><hr>
Площадь: <input type="text" name="res" size=10>
</FORM>
```

В этой форме шесть элементов. Первый элемент служит для ввода строки текста. Следующие три элемента образуют группу и являются переключателями. Пятый элемент создает кнопку сброса, нажатие которой отменяет все сделанные изменения. Шестой элемент является элементом для ввода строки.

Так как объект forms имеет свойство-массив elements, в котором содержатся ссылки на элементы формы в порядке их перечисления в теге <FORM>, то получить доступ к первому элементу формы можно либо с помощью значения параметра name этого элемента (document.form1.data), либо используя объектную модель JavaScript (document.forms[0].elements[0]). Второй элемент рассматриваемой формы можно получить, если воспользоваться конструкцией document.forms[0].elements[1]. Это элемент-переключатель, определенный в составе группы элементов. В рассматриваемом примере группа элементов состоит из трех переключателей. В одну группу входят элементы с одинаковым значением параметра name. Доступ к следующим элементам группы может быть осуществлен так: document.forms[0].elements[2], document.forms[0].elements [3]. Обязательный параметр value должен иметь уникальное значение для каждого элемента группы. Пользователь может выбрать только один вариант.

Напишем сценарий, в котором в зависимости от длины стороны или радиуса и формы выбранной фигуры вычисляется ее площадь. Для простоты будем считать, что фигура может иметь либо форму квадрата (задается его

сторона), либо форму круга (задается радиус), либо форму равностороннего треугольника (задается его сторона).

Площадь рассматриваемых фигур считается по формуле ka^2, где k - коэффициент, зависящий от формы выбранной фигуры; a - задаваемое пользователем значение. Вычисления будут проще, если коэффициент k указать в качестве значения параметра value соответствующего переключателя. Щелчок на элементе "переключатель" соответствует событию click, обработка которого заключается в вызове функции test. Функция имеет единственный параметр, значение параметра - value переключателя, которое служит для вычисления площади фигуры.

HTML-код приведен в листинге 1.

Листинг 1. Вычисление площади выбранной с помощью переключателя фигуры

```
<HTML>
<HEAD>
<TITLE>Данные из формы типа "переключатель". Событие Click </TITLE>
<script language="JavaScript">
<!--//
function test (k)
{ var a= form1.data.value
if (a !="" )
form1.res.value= k*Math.pow(a,2)
else alert ("Введите значение")
}
//-->
</script>
</HEAD>
<BODY>
<FORM name="form1">
Введите значение
<input type="text" name="data" size=10>
<hr>
Укажите форму <br>
<input type="radio" name="fv" value=1
onClick="test(form1.elements[1].value)">квадрат<br>
<input type="radio" name="fv" value=3.14
onClick="test(form1.elements[2].value)">круг<br>
<input type="radio" name="fv" value=0.42
onClick="test(form1.elements[3].value)">треугольник<hr>
<input type="reset" value="0тменить"><hr>
Площадь: <input type="text" name="res" size=10>
</FORM>
```

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

Пример 2. Выбор параметров обтекания изображения текстом

Напишем сценарий, который предоставляет возможность пользователю задавать значения параметров, определяющих, к какому полю окна (левому или правому) прижимается изображение, и, соответственно, с какой стороны текст его обтекает.

Если значение параметра align равно Left, то изображение прижимается к левому краю окна просмотра браузера, а текст или другие элементы документа "обтекают" изображение с правой стороны. Текст, размещаемый рядом с изображением, может занимать несколько строк. По умолчанию значение параметра align равно Left. При нажатии на кнопку Обновить для изображения и текста будут установлены значения параметров, принимаемых по умолчанию.

Пример HTML-кода, который содержит сценарий, обеспечивающий выполнение действий, задаваемых пользователем, приведен в листинге 2.

Листинг 2. Обтекание текстом изображения

```
<HTML>
<HEAD>
<TITLE>Изображение и текст. Обтекание</TITLE>
<script>
<!--
function chpict(obj)
{
if ((obj.elements[0]).checked)
document.mypict.align="Left"
else
document.mypict.align="Right"
}
function rset(obj)
{document.mypict.align="Left"}
//-->
</script>
</HEAD>
<BODY>
<CENTER>
<H4>Изображение и текст.
Обтекание изображения текстом</H4>
</CENTER>
<FORM name="form1">
```

```
Выберите значение параметра выравнивания нажмите кнопку <B>Просмотр</B>.<br>
 <PRE>
 <input type="radio" name="alg" checked value=ld>(left) изображение выравнивается по левому краю
 <input type="radio" name="alg" value=rd>(right) изображение выравнивается по правому краю
 </PRE>
 <input type="button" value= "Просмотр" onclick="chpict(form1)">
 <input type="reset" value="Отменить" onclick="rset(form1)"> </FORM>
 <TABLE bgcolor="F8F8FF">
 <TR><TD>Иван Иванович Шишкин является одним из основоположников
 русского национального пейзажа.
 <IMG src=pl.jpg name=mypict align=left border=3 width=310>
 В полотне "Рожь" Шишкин создал образ большой эпической силы
 и подлинно монументального звучания. Могучая, полная
 богатырских сил природа, богатый, привольный край. (Т. Юрова)
 </TD></TR>
 </TABLE>
 </BODY>
 </HTML>
```

Если изображение рассматривается как элемент строки, то значения параметров выравнивания задают расположение изображения относительно строки текста. Верхняя граница изображения может быть выровнена либо по самому высокому текстовому элементу текущей строки, либо по самому высокому элементу в строке (например, другому изображению). Базовой считается нижняя часть линии текста, которая проводится без учета нижней части некоторых символов. Середину изображения можно выровнять либо по базовой линии, либо по середине текущей строки. Нижнюю часть изображения можно выровнять по базовой линии, либо по нижней границе текущей строки.

Задания

1. Проверить примеры из лабораторной работы.
2. Напишите сценарий, который позволяет продемонстрировать изменения размеров и положения на странице горизонтальной линии.
3. Разработайте анкету, определяющую пол, возраст, семейное положение и т.п., человека.

Лабораторная работа №6
Флажки

Элемент управления "флажок" используется в случае, когда из предложенных вариантов можно выбрать как один, так и несколько. Каждый вариант выбора задается флажком, который можно либо установить, либо сбросить. Флажок определяется в теге <input> значением checkbox параметра type. Обязательным параметром является параметр value, значение которого будет передано на обработку в случае выбора нажатием кнопки.

Пример 1. Выбор характеристик издания

Предположим, читателю предлагается заполнить анкету, в которой требуется указать название любимого издания и выбрать из предложенного списка характеристики, которые присущи рассматриваемому изданию.

Для задания характеристик издания можно воспользоваться флажком. Пользователь устанавливает флажки для тех свойств, которыми, по его мнению, обладает издание. Обработка анкеты будет состоять в том, что выбранные свойства будут отражены в поле ввода многострочного текста.

При щелчке мышью по флажку возникает событие click, обработка которого состоит в вызове функции set с одним параметром, принимающим значение параметра value флажка. Для формирования строки результата служит глобальная переменная s; к имеющемуся значению добавляется значение параметра функции и помещается в текстовое поле. Если нажать ни кнопку Отмена, то очистятся все поля формы. Однако следует позаботиться о том, чтобы значение переменной s изменилось на начальное. Значение параметра реакции на событие click при щелчке по кнопке Отмена задается оператором присваивания, обеспечивающим начальные условия.

HTML-код представлен в листинге 1.

Листинг 1. Анкета читателя

```
<HTML>
<HEAD>
<TITLE>Анкета читателя</TITLE>
<script>
<!--
var s="Вас привлекает: \r\n"
function set(vch)
{ s=s+vch + "\r\n"; document.form1.area.value=s }
//-->
</script>
</HEAD>
<BODY bgcolor="F8F8FF">
<CENTER>
```

```
<H3 align="center">Анкета читателя</H3>
<FORM name="form0">
<H4>Введите название любимого журнала или газеты</H4>
<input type="text" name="n1" size=45><br>
</FORM>
<FORM name="form1">
<H4>Что Вас привлекает в издании?</H4>
<TABLE border=3 align=center> <TR>
<TD><img src="purpsm.gif" align="center"></TD>
<TD><input type="checkbox" name="m1" value="Стиль подачи материала"
onClick="set(form1.elements[0].value)">
Стиль подачи материала<br>
<input type="checkbox" name="m2" value="Достоверность информации"
onClick="set(form1.elements[1].value)">
Достоверность информации<br>
<input type="checkbox" name="m3" value="Дизайн и оформление"
onClick="set(form1.elements[2].value)">
Дизайн и оформление<br>
</TD></TR></TABLE>
<textarea name="area" cols=35 rows=7> </textarea><br>
<input type="reset" value="Отмена"
onclick= "s='Вас привлекает: \r\n'">
</FORM>
</BODY>
</HTML>
```

В рассмотренных примерах значения параметра name флажков были различны, поскольку каждый флажок существовал независимо от других. Флажки можно объединить в группу. Для этого следует всем флажкам присвоить одно и то же значение параметра name.

Пример 2. Использование флажков в анкете переводчика

В анкете требуется указать те языки, которыми владеет переводчик. Предположим, что за знание каждого языка назначается определенная сумма. Размер вознаграждения определяется после заполнения анкеты в зависимости от тех языков, которыми пользователь владеет. По результатам заполненной переводчиком анкеты напишите сценарий определения размера вознаграждения.

Для задания сведений о том, владеет ли пользователь определенным языком, удобно применять флажок. При щелчке мышью по кнопке Вознаграждение выполняется функция grant(). Требуется проанализировать состояние флажков. Свойство checked возвращает логическое значение, представляющее текущее значение отдельного флажка (true или false). Воспользуемся тем, что каждый объект form имеет свойство-массив elements,

получим доступ к каждому флажку формы. Состояние первого флажка можно определить с помощью следующей конструкции:

(document.forml.elements[0]).checked

второго -

(document. form1.elements[1]).checked

и т. д. В переменной к накапливается сумма. Шаг увеличения этой переменной задается в качестве значения параметра value. После анализа всех флажков полученная сумма выводится в документ.

HTML-код представлен в листинге 2.

Листинг 2. Данные, представленные флажком. Анкета переводчика

```
<HTML>
<HEAD>
<TITLE>Данные, представленные флажком. Анкета переводчика</TITLE>
<script language="JavaScript">
<!-- //
function grant()
{ var d= document
var k=0;
if ((d.form1.elements[0]).checked)
k=k+Number(d.form1.elements[0].value)
if ((d.form1.elements[1]).checked)
k=k+Number(d.form1.elements[1].value)
if ((d.form1.elements[2]).checked)
k=k+Number(d.form1.elements[2].value)
form1.ww.value="Вам полагается вознаграждение "+k+" у.е."
}
//-->
</script>
</HEAD>
<BODY>
<H3>Анкета для переводчиков</H3>
Укажите те языки, которыми Вы владеете в совершенстве: <br>
<FORM name="form1">
<input type="checkbox" name="lan" value=100>русский<br>
<input type="checkbox" name="lan" value=200>английский<br>
<input type="checkbox" name="lan" value=300>французский<br>
<input type="button" value=Вознаграждениие onClick="grant()"> <hr>
<input type="Text" size=50 name="ww" value=""><br>
<input type="reset" value="Отменить">
</FORM><hr>
</BODY>
</HTML>
```

Упражнение

Напишите сценарий обработки анкеты слушателя курсов. Пользователь может выбрать курс, его продолжительность, язык, на котором он готов работать с преподавателем, и форму отчетности. В зависимости от этих параметров определяется стоимость отдельного курса и стоимость всего обучения. Анкета приведена на рис. 2.

Лабораторная работа №7
Списки

Если элементов много, то представление их с помощью флажков или переключателей увеличивает размер формы. В этом случае варианты выбора могут быть представлены в окне браузера более компактно с помощью тега <select>. Тег имеет несколько параметров. Параметр name является обязательным. Для того чтобы установить число одновременно видимых элементов, следует задать параметр size=n. Когда n равно 1, то отображается ниспадающее меню или список выбора; при n>1 выводится список с n одновременно видимыми значениями. Если параметр size не задан, то по умолчанию принимается значение равное 1. Указание параметра multiple означает, что из меню или списка можно выбрать несколько элементов. Элементы меню задаются внутри тега <select> с помощью тега <option>. Общий вид тега таков:

<option selected value=строка>

Параметр selected означает, что данный элемент списка считается выбранным по умолчанию. Параметр value содержит значение, которое передается, если данный элемент выбран из списка или меню.

Пример 1. Обработка анкеты переводчика

Напишем сценарий обработки анкеты переводчика. Сведения о тех языках, которыми владеет переводчик, требуется задать с помощью списка, Выбранные языки следует помещать в поле ввода многострочного текста.

Напомним, что событие change происходит в тот момент, когда значение элемента формы text, select или textarea изменилось, и элемент потерял фокус. Будем обрабатывать анкету переводчика следующим образом. Параметр обработки события поместим в тег <select>. Как только выбран конкретный язык, т. е. произошло событие change, выполняется функция gr:

```
function gr(obj,m)
 { var r=100*(Number(((obj.elements[0])[m]).value)+1)
 s+=((obj.elements[0]) [m] ).text+"\r\n"
 obj.restext.value=s
 sum+=r
 obj.res.value=r}
```

Первый параметр - имя формы, второй - значение параметра value выбранного элемента. Второй оператор обеспечивает формирование строки всех выбранных пользователем языков. Третий оператор помещает вычисленное значение в текстовое поле. В результате выполнения четвертого оператора присваивания в переменной sum формируется значение, которое, затем при нажатии кнопки Сумма будет выведено в текстовое поле.

Последний оператор помещает значение для выбранного языка в соответствующее поле формы. Полностью документ со сценарием и формами может быть описан так, как указано в листинге 1.

Листинг 1. Реакция на событие change в теге <select>

```
<HTML>
<HEAD>
<TITLE>Реакция на событие Change в теге select</TITLE>
<script language="JavaScript">
<!-- //
var s=""
var sum=0
function gr(obj,m)
{ var r=100*(Number(((obj.elements[0])[m]).value)+1)
s+=((obj.elements[0])[m]).text+"\r\n"
obj.restext.value=s
sum+=r
obj.res.value=r
}
//-->
</script>
</HEAD>
<BODY>
<FORM name="form1">
<H3>Анкета переводчика</H3>
<TABLE border=3 bgcolor=silver>
<TR><TH>Выбранный язык</TH><TH>Результат</TH></TR>
<TR>
<TD valign=top>
<select name="data" size=3 onChange="gr(form1,form1.data.value)">
<option value=0>русский
<option value=1>английский
<option value=2>французский
</select><P>
<input type="text" name="res" size=15>
</TD>
<TD><TEXTAREA name="restext" cols=15 rows=6>
</TEXTAREA><P>
<input type="button" value=Сумма onClick="form1.resgr.value=sum">
<input type="text" name="resgr" size=10>
</TD></TR></TABLE><p>
<input type="reset" value="Отменить" onClick="sum=0; s="">
</FORM>
</BODY>
```

```
</HTML>
```

Пример 2. Тест "Города и памятники"

Напишем сценарий обработки теста "Города и памятники". Названия городов и памятников задаются с помощью списков. Пользователь выбирает в левом перечне название города, а в правом - памятник, расположенный в этом городе. После нажатия кнопки Результат в текстовое поле выводится количество правильных ответов.

В сценарии используются три глобальные переменные. Переменная q хранит последнее выбранное значение в левом столбце; переменная a - выбранное значение правого столбца; значение переменной sum содержит число правильных ответов. В двух списках для правильной пары "вопрос/ответ" совпадают соответствующие значения параметра value. Эти значения проверяются после выбора элемента списка правого столбца. Результат тестирования можно посмотреть, если нажать кнопку Результат.

Сценарий, реализующий простую обработку теста, представлен в листинге 2.

Листинг 2. Простая тестирующая программа

```
<HTML>
<HEAD>
<TITLE>Города и памятники.</TITLE>
<script>
<!--
var sum=0
var q
var a
function eq()
{ q=test.question.value
a=test.answer.value;
if (a==q) sum +=1
}
function result()
{ document.test.res.value=sum}
//-->
</script>
<BODY background="fon3.gif">
<h3 align=center>Города и памятники</h3>
<FORM name="test">
<TABLE border=3 align=center cellpadding=5
cellspacing=6 bgcolor= silver>
<TR><TH>Памятник</TH><TH>Находится в городе</TH>
<TR><TD>
```

```
<select size =7 name="question"
onChange="q=test.question.value">
<option value="mad">Музей Прадо<br>
<option value="ber" >Рейхстаг<br>
<option value="mil">Оперный театр Ла Скала<br>
<option value="ier">Стена Плача<br>
<option value="mek">Священный камень Кааб<br>
<option value="spb">Медный Всадник<br>
<option value="mos">Третьяковская галерея<br>
<option value="par">Триумфальная Арка<br>
<option value="new">Статуя Свободы<br>
<option value="lon">Тауэр<br>
</select>
</TD>
<TD>
<select size=7 name="answer" onChange="eq()">
<option value="spb">Санкт-петербург<br>
<option value="mos">Москва<br>
<option value="mek">Мекка<br>
<option value="ier">Иерусалим<br>
<option value="mil">Милан<br>
<option value="par">Париж<br>
<option value="mad">Мадрид<br>
<option value="lon">Лондон<br>
<option value="new">Нью-Йорк<br>
<option value="ber">Берлин<br>
</select>
</TD></TR>
</TABLE>
<CENTER><P>
<input type="button" value="Результат" onclick="result()"><br>
Количество правильных ответов
<input type="text" name="res" size="5"><P>
<input type="reset" value= "Обновить" onclick="sum=0">
</FORM>
</BODY>
</HTML>
```

Задания

1. Проверить примеры из лабораторной работы.

2. Напишите сценарий, который позволяет выбрать для таблицы и составляющих ее ячеек либо цвет фона, либо фоновое изображение, либо и то и другое. Предусмотрите возможность задания своего цвета фона для каждой ячейки.

3. Напишите сценарий, который позволяет посчитать стоимость предполагаемой покупки. Задается список продуктов, цена за единицу товара и количество экземпляров.

Лабораторная работа №8
Фреймы

Окно просмотра браузера можно разбить на несколько прямоугольных областей, называемых фреймами. Области соприкасаются друг с другом и в каждую из областей можно загрузить отдельный HTML-документ, и работать с ним независимо от документов, загруженных в другие области окна или фрейма. Между фреймами можно организовать взаимодействие, например, выбор ссылки в одном из фреймов позволит изменить содержимое других фреймов. Фреймы часто используются в случаях, когда возникает необходимость загрузить документ в одну из областей при работе в другой области, или когда следует отобразить информацию, которая должна постоянно находиться на экране.

Пример 1. Простая фреймовая структура

Создадим документ, который разбивает область экрана на две части. Левая часть содержит оглавление разделов документа, который располагается в правой части. При выборе пункта оглавления в левой части появляется соответствующий раздел документа в правой части.

Разобьем область экрана на два фрейма. Левый фрейм занимает 25% ширины всего окна и будет содержать оглавление разделов документа, который загрузим в правый фрейм. Пусть имя файла, содержащего оглавление - contents.htm, а имя документа - ch.htm. Фреймовая структура задает способ организации экрана и определяет, какие документы должны быть первоначально загружены во фреймы. Создать описанную фреймовую структуру можно, если использовать документ, содержащий HTML-код, представленный в листинге 1.

Листинг 1. Создание простой фреймовой структуры

```
<HTML>
<HEAD>
<TITLE>Простая фреймовая структура</TITLE>
</HEAD>
<frameset cols="25%,75%">
<frame src=contents0.htm name=left>
<frame src= ch.htm name=right>
</frameset>
</HTML>
```

Параметр cols тега <frameset> имеет вид cols="список значений". В списке через запятую перечисляются значения, которые определяют размеры фреймов. Список должен содержать не менее двух значений. Значения могут задаваться в процентах, в пикселах, в относительных единицах.

Тег <frame> определяет один фрейм. Он должен располагаться внутри парного тега <frameset> и </frameset>. Число тегов <frame> должно совпадать с количеством тегов, определенных при описании фреймовой структуры. В рассматриваемом примере в теге <frameset cols="25%,75%"> определено два фрейма, поэтому в дальнейшем следует описание каждого из фреймов с помощью тега <frame>.

Значение параметра src тега <frame> определяет адрес документа, который первоначально загружается во фрейм. В рассматриваемом случае в левый фрейм загружается документ с именем contents0.htm, а в правый фрейм - документ с именем ch.htm. В теге параметр name определяет имя фрейма, необходимое для указания, в какой фрейм загрузить документ. Если имя фрейма не задавать, то будет создан фрейм без имени, но сослаться на него из других фреймов будет нельзя.

Пример 2. Фреймовая структура с загружаемыми документами

Создадим документ, левая часть которого представляет оглавление, а в правую часть загружаются документы выбранного пункта оглавления. Документы, соответствующие пунктам оглавления, хранятся в разных файлах.

При решении задачи экран по-прежнему разбивается на два фрейма. Левый фрейм занимает 30% ширины всего окна и будет содержать оглавление документов, которые могут быть просмотрены пользователем при выборе соответствующего пункта. Правый фрейм занимает большую часть окна просмотра и предназначен для отображения самих документов. При первоначальной загрузке оба фрейма делят окно просмотра по вертикали в соотношении 30% и 70%. Данное соотношение может меняться при просмотре. Каждый из фреймов имеет свою полосу прокрутки, обеспечивающую просмотр всего документа. При выборе ссылки в левом фрейме соответствующий документ будет загружен в правый фрейм. Такая структура позволяет одновременно видеть на экране и оглавление документов, и сами документы.

Пусть оглавление документа содержит шесть пунктов и располагается в файле с именем contents.htm. Требуется, чтобы файл, содержащий оглавление, загружался в левый фрейм. Файлы с именами ch1.htm, ch2.htm, ..., ch6.htm содержат документы, соответствующие пунктам оглавления.

Фреймовая структура мало отличается от той, какая была рассмотрена в предыдущем примере (листинг 2, а).

Листинг 2, а. Задание фреймовой структуры

```
<HTML>
<HEAD>
<TITLE>Простая фреймовая структура</TITLE>
</HEAD> <frameset cols="30%,70%">
```

```
<frame src=contents.htm name=left>
<frame src=empty.htm name=right>
</frameset>
</HTML>
```

В правый фрейм первоначально загружается файл с именем empty.htm. Если сразу неизвестно, какой файл загружать во фрейм, то можно использовать файл, содержащий HTML-код (листинг 2, б).

Листинг 2, б. Документ для первоначальной загрузки

```
<HTML>
<HEAD>
<TITLE>Пустой документ</TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

В левый фрейм помещается Оглавление, которое содержит ссылки на документы, расположенные в различных файлах. Оглавление может быть сформировано так, как указано в листинге 2, в.

Листинг 2, в. Оглавление, загружаемое в левый фрейм

```
<HTML>
<HEAD>
<TITLE>Оглавление</TITLE>
</HEAD>
<BODY background="decor.gif" bgcolor=silver>
<base target=right>
<h3>Оглавление</h3>
<OL>
<LI><A href="ch1.htm">Основы языка HTML </A>
<LI><A href="ch2.htm">Графика </A>
<LI><A href="ch3.htm">Изображение-карта </A>
<LI><A href="ch4.htm">Списки </A>
<LI><A href="ch5.htm">Таблицы </A>
<LI><A href="ch6.htm">Фреймы </A>
</OL>
</BODY>
</HTML>
```

Пример 3. Обмен содержимым фреймов

Создадим документ, разбивающий окно просмотра с помощью фреймов на три прямоугольные области. Верхняя область содержит два фрейма, нижняя область состоит из одного фрейма. В нижней области располагается кнопка, которая обменивает содержимое верхних фреймов.

Область окна просмотра сначала разбивается на два фрейма по горизонтали. Параметр rows="список значений" задает количество и размеры фреймов по горизонтали. При описании тега <frameset rows="*,50"> определяется, что размер нижнего фрейма 50 пикселов, остальное пространство отводится под верхний фрейм. Верхний фрейм в свою очередь делится по вертикали на два фрейма, размеры которых заданы в процентах к области просмотра. HTML-код документа, создающий описанную фреймовую структуру, выглядит так, как указано в листинге 3, а.

Листинг 3, а. Три фрейма с кнопкой для обмена содержимого фреймов

```
<HTML>
<HEAD>
<TITLE>Три фрейма с кнопкой для обмена</TITLE>
</HEAD>
<frameset rows="*,50">
<frameset cols="55%,45%">
<frame src=lpict.htm name=left>
<frame src=rtext.htm name=right>
</frameset>
<frame src=but1.htm name=butt>
</frameset>
</HTML>
```

Для ссылки на фреймы будем, как и ранее, использовать имена. Сначала следует запомнить имя файла, загруженного в левый фрейм. Для этого используется переменная l. Затем в левый фрейм загружается документ, расположенный в правом фрейме. Это достигается выполнением оператора присваивания

top.left.location=top.right.location

И, наконец, в правый фрейм загружается тот документ, адрес которого запомнили в переменной l.

HTML-код для нижнего фрейма представлен в листинге 3, б.

Листинг 3, б. Сценарий для нижнего фрейма

```
<HTML>
<HEAD>
<TITLE>Кнопка для смены содержимого фреймов</TITLE>
```

```
<script>
<!--//
function chframe()
{ var l=top.left.location
top.left.location=top.right.location
top.right.location=l
}
//-->
</script>
</HEAD>
<BODY background="decor.gif" bgcolor=silver>
<CENTER>
<FORM name=form1>
<input type="button" value=Обмен onclick="chframe()">
</FORM>
</CENTER>
</BODY>
</HTML>
```

При щелчке по кнопке Обмен содержимое верхних фреймов поменяется местами.

Функция, осуществляющая обмен содержимым фреймов, может быть описана следующим образом:

```
function chframe()
 { var l=top.frames[0].location
 top.frames[0].location=top.frames[1].location
 top.frames [1].location=l
 }
```

Браузер Microsoft Internet Explorer разрешает использование так называемых плавающих фреймов, описание которых может быть расположено в тексте обычного HTML-документа. Для определения плавающего фрейма используется тег <iframe>. В указанном теге могут встречаться те же параметры, что и в теге описания обычных фреймов, кроме параметра noresize, т. к. размер плавающего фрейма не может быть изменен пользователем. Браузеры, не поддерживающие тег <iframe>, отображают записанную между тегами <iframe> и </iframe> информацию.

Для плавающих фреймов с помощью параметров можно задавать размеры фрейма, горизонтальное выравнивание, размер отступа содержимого фрейма от границ. В следующем примере приведено описание плавающего фрейма и заданы параметры. Фрейм имеет высоту 320 пикселов, занимает по ширине 60% окна, расположен справа от текста, полосы прокрутки будут установлены в случае, если документ не станет помещаться во фрейм. Содержимое фрейма должно быть отделено от границы по

горизонтали и вертикали на заданное число пикселов.

<iframe src=pictsh.htm name="flframe" height=320 width=60% hspace=50 vspace=10
scrolling=auto align=right>

Задания

1. Создайте документ, разбивающий окно просмотра с помощью фреймов на две прямоугольные области: верхнюю и нижнюю. В верхней области поместите оглавление в виде списка, при выборе пунктов которого соответствующий раздел должен появляться в нижней части окна.
2. Создайте документ, разбивающий окно просмотра с помощью фреймов на две прямоугольные области: левую и правую. В левой области поместите оглавление, при выборе пунктов которого соответствующий пункту раздел должен появляться в правой части окна. Оглавление представьте с помощью графического вертикального меню.

Лабораторная работа №9
Повторяющиеся вычисления - циклы

Для успешного решения широкого круга задач требуется многократно повторить некоторую последовательность действий, записанную в программе один раз. В том случае, когда число повторений последовательности действий нам неизвестно, либо число повторений зависит от некоторых условий, можно воспользоваться оператором цикла вида:

while (B) {s}

где B - выражение логического типа; s - операторы, называемые телом цикла. Операторы s в фигурных скобках выполняются до тех пор, пока условие B не станет ложным.

Пример 1. Нахождение общего делителя

Напишем программу, которая для двух заданных чисел определяет наибольший общий делитель.

При решении задачи воспользуемся алгоритмом Евклида. Если значение m равно нулю, то наибольший общий делитель чисел n и m равен n:

НОД(n, 0) = n.

В остальных случаях верно следующее соотношение:

НОД(n, m) = НОД(m, n%m).

В функции nod переменная p используется для получения остатка от деления чисел n и m (листинг 1). Выполнение цикла продолжается до тех пор, пока значение p не станет равным нулю. Последнее вычисленное значение m равно наибольшему общему делителю.

Листинг 1. Наибольший общий делитель двух чисел

```
<HTML>
<HEAD>
<TITLE>Наибольший общий делитель двух чисел</TITLE>
<script language="JavaScript">
<!-- //
function nod(obj)
{ var n=obj.num1.value
var m=obj.num2.value
var p = n%m
while (p!=0)
{ n=m
m=p
p=n%m
}
```

```
obj.res.value=m
}
//-->
</script>
</HEAD>
<BODY>
Наибольший общий делитель двух заданных чисел
<FORM name="form1">
Введите число <input type="text" name="num1" size="8"><br>
Введите число <input type="text" name="num2" size="8"><br>
<input type="button" value="Вычислить" onClick="nod(form1)"><br>
Наибольший общий делитель <input type="text" name="res"
size="8"><hr>
<input type="reset" value="Отменить">
</FORM>
</BODY>
</HTML>
```

Если число повторений заранее известно, то можно воспользоваться следующим оператором цикла, который часто называют оператором цикла арифметического типа. Синтаксис этого оператора таков:

for (A; B; I){S}

Выражение A служит для инициализации параметра цикла и вычисляется один раз в начале выполнения цикла. Выражение B (условие продолжения) управляет работой цикла. Если значение выражения ложно, то выполнение цикла завершается, если истинно, то выполняется оператор S, составляющий тело цикла. Выражение I служит для изменения значения параметра цикла. После выполнения тела цикла S вычисляется значение выражения I, затем опять вычисляется значение выражения B и т.д. Цикл может прекратить свою работу в результате выполнения оператора break в теле цикла.

Пример 2. Совершенные числа

Напишем программу, определяющую, является ли заданное число n совершенным.

Совершенным называется число n, равное сумме своих делителей, не считая самого числа. Например, число 6 является совершенным, т. к. верно 6 = 1 + 2 + 3, где 1, 2, 3 - делители числа 6. Число 28 также является совершенным, справедливо равенство 28 = 1 + 2 + 4 + 7 + 14. При решении задачи воспользуемся только функцией sumdei (листинг 2).

Листинг 2. Итерационные методы. Совершенные числа

```
<HTML>
<HEAD>
<TITLE>Итерационные методы. Совершенные числа</TITLE>
<script language="JavaScript">
<!-- //
function sumdel(n)
{ var s=1;
for (var i=2; i<=n/2; i++)
{ if (n % i == 0) s += i }
 return s
}
function sov(obj)
{ var n=obj.numb.value;
var s=""
if (n==sumdel(n)) s="совершенное"
else s="не является совершенным"
return s
}
//-->
</script>
</HEAD>
<BODY>
<P> Итерационные методы. Совершенные числа</P>
<FORM name="form0">
Введите натуральное число: <input type="text" size=8 name="numb">
<input type="button" value=Выполнить
onClick="this.form.res.value=sov(form0)"><hr>
Данное число: <input type="text" size=24 name="res"><hr>
<input type="reset" value=Отменить>
</FORM>
</BODY>
</HTML>
```

Обратите внимание на значение параметра обработки события. В данном случае это оператор присваивания, в правой части которого вызов функции sov.

Оператор for...in используется для анализа свойств объекта. Синтаксис оператора:

for (i in t) {s}

где i - переменная цикла; t - объект; s - последовательность операторов.

В результате выполнения оператора цикла производится перебор свойств объекта. Переменная цикла при каждом повторении содержит значение свойства объекта. Количество повторений тела цикла s равно числу свойств, определенных для объекта t.

Пример 3. Определение свойств элемента формы

Напишем сценарий, с помощью которого можно определить свойства элемента формы "поле ввода многострочного текста".

Свойства объекта с помощью оператора цикла формируются в строке result, затем после просмотра всех свойств значение строки result помещается в поле ввода многострочного текста.

Сценарий определения свойств текстового поля приведен в листинге 3.

Листинг 3. Операции над объектами. Свойства текстового поля

```
<HTML>
<HEAD>
<TITLE>Операции над объектами. Свойства текстового поля</TITLE>
<script language="JavaScript">
<!-- //
function propobj (obj)
{ var result = ""
for (var i in obj)
{ result += obj.data.value +"." +i+ " = " +(obj.data)[i]+"\r\n"}
result += "\n\r"
form1.data.value=result
}
//-->
</script>
</HEAD>
<BODY bgcolor=F8F8FF>
<CENTER>
<H4>Определение свойств объектов</H4>
<FORM name="form1">
<input type="button" value=Выполнить onClick="propobj (form1)"><hr>
<textarea name="data" cols=30 rows=10 id=1>Текст</textarea><hr>
<input type="reset" value=Очистить>
</CENTER>
</FORM>
</BODY>
</HTML>
```

Задания

1. Проверить примеры из лабораторной работы.
2. Напишите программу, которая "переворачивает" заданное натуральное число.

3. Напишите сценарий, в котором определяется количество "счастливых" шестизначных автобусных билетов, т. е. таких, в номерах которых сумма первых трех цифр равна сумме трех последних.

4. Напишите программу, определяющую все делители заданного натурального числа.

Лабораторная работа №10
Обработка и представление дат

Встроенный объект Data применяется для представления и обработки даты и времени. Он не имеет свойств, но обладает несколькими методами, позволяющими устанавливать и изменять дату и время. В языке JavaScript дата определяется числом миллисекунд, прошедших с 1 января 1970 года.

Объект Data создается оператором new с помощью конструктора Data. Если в конструкторе отсутствуют параметры, то значением new Data () будет текущая дата и время. Значением переменной my_data1, определенной следующим образом:

var my_data1 = new Data()

будет объект, соответствующий текущей дате и времени.

Параметром конструктора new Data может быть строка формата "месяц, день, год часы: минуты: секунды". Опишем переменную my_data2 и присвоим ей начальное значение:

var my_data2 = new DataC'Fv, 12, 1978 16:45:10")

Переменная my_data2 определяет дату 12 февраля 1978 года и время 16 часов 45 минут и 10 секунд. Значения часов, минут, секунд можно опустить, в этом случае они будут равны нулю:

var my_data3 = new Data("Feb, 12, 1978")

Параметры конструктора new Data могут определять год, месяц, число, время, минуты, секунды с помощью чисел. Дату 12 февраля 1978 года и время 16 часов 45 минут и 10 секунд можно задать так:

var my_data4 = new Data(78, 1, 12, 16, 45, 10)

Если время опустить, то описание будет следующим:

var my_data5 = new Data(78, 1, 12)

Все числовые представления даты нумеруются с нуля, кроме номера дня в месяце. Месяцы представляются числами от 0 (январь) до 11(декабрь), поэтому второй параметр при задании переменных my_data4 и my_data5 равен 1.

Методами объекта Date можно получать и устанавливать отдельно значения месяца, дня недели, часов, минут и др.

-Метод getDate возвращает число в диапазоне от 1 до 31, представляющее число месяца.

-Метод getHours возвращает час суток. Значение возвращается в 24-часовом формате от 0 (полночь) до 23.

-Метод getMinutes возвращает минуты как целое от 0 до 59.

-Метод getseconds возвращает число секунд как целое от 0 до 59.

-Метод getDay возвращает день недели как целое число от 0 (воскресенье) до 6 (суббота).

-Метод getMonth возвращает номер месяца в году как целое число в интервале между 0 (январь) и 11 (декабрь). Обратите внимание, что номер месяца не соответствует стандартному способу нумерации месяцев.

-Метод getYear выдает год объекта.

В следующем примере эти методы используются для формирования текущего времени.

Пример 1. Определение текущего времени

Напишем сценарий, который определяет текущее время и выводит его в текстовое поле в формате "чч:мм:сс".

В переменной res формируется строка, которая затем будет отображена в поле rest формы с именем form1. Для того чтобы уточнить время, следует еще раз нажать кнопку Время и т. д. Полностью сценарий приведен в листинге 1.

Листинг 1. Определение времени

```
<HTML>
<HEAD>
<TITLE>Определение времени</TITLE>
<script language="JavaScript">
<!-- //
function c1()
{ var d=document
var t=new Date()
var h=t.getHours()
var m=t.getMinutes()
var s=t.getseconds()
var res=""
if (h < 10)
res += "0" + h
else res += h
if (m < 10) res += ":0"+m
else res += ":"+m
if (s < 10) res += ":0"+s
else res += ":"+s
d.form1.rest.value = res
}
//-->
</script>
</HEAD>
```

```
<BODY bgcolor="#FFFFCC">
<CENTER>
<IMG src=alarmWHT.gif><br>
При нажатии кнопки <B>Время</B>, Вы узнаете, который час
<FORM name="form1">
<input type="button" value=Время onClick="c1()">
<input type="text" size=10 narae="rest"><br>
</FORM>
</BODY>
</HTML>
```

Можно сделать так, что через некоторый заданный период значение времени будет обновляться. Для этого можно использовать функцию setTimeout ("c1()", 3000). Функция setTimeout выполняет указанные в первом параметре действия по истечении интервала времени, задаваемого вторым параметром. В приведенном примере через три секунды будет снова осуществлен вызов функции c1.

Перечисленные ниже методы позволяют устанавливать различные значения для объекта Date.

-Метод setYear устанавливает значение года для объекта Date.

-Метод setDate устанавливает день месяца. Параметр должен быть числом в диапазоне от 1 до 31.

-Метод setMonth устанавливает значение месяца. Параметр должен быть числом в диапазоне от 0 (январь) до 11 (декабрь).

-Метод setHours устанавливает час для текущего времени, использует целое число от 0 (полночь) до 23 для установки даты по 24-часовой шкале.

-Метод setMinuts устанавливает минуты для текущего времени, использует целое число от 0 до 59.

-Метод setSeconds устанавливает секунды для текущего времени, использует целое число от 0 до 59.

-Метод setTime устанавливает значение объекта Date и возвращает количество миллисекунд, прошедших с 1 января 1970 года.

Пример 2. Пятница 13

Напишем сценарий, с помощью которого определяются все даты в указанном году, приходящиеся на пятницу, 13 число.

При написании сценария будем поступать следующим образом: перебирать с помощью цикла месяцы и в каждом месяце устанавливать номер дня 13. Установка требуемой даты выполняется использованием методов работы с датой:

```
t.setYear (y)
 t.setMonth (i)
 t.setDate (13)
```

Далее следует проверить, какой номер дня соответствует этой дате. Если номер равен пяти ((t. getoay ()) ==5), то день недели - пятница, найденный месяц следует запомнить. Для формирования ответа используется строковая переменная c, после запоминания названия месяца добавляется символ перевода строки. HTML-код со сценарием приведен в листинге 2.

Листинг 2. В какие месяцы года 13 число попадает на пятницу?

```
<HTML>
<HEAD>
<TITLE>В какие месяцы года 13 число попадает на пятницу?</TITLE>
<script language="JavaScript">
<!-- //
function def13(obj)
{ var t= new Date()
var c=""
var y=Number(obj.fye.value)
for (var 1=0; i <=11; i++)
{ t.setYear(y)
t.setMonth(i)
t.setDate(13)
if ( (t.getDay())==5)
c = c+ fmon(i)+ "\r\n"
}
obj.res.value = c
}
function fmon(mont)
{
var s
switch (mont)
{ case 0 s="январь"; break;
case 1 s="февраль"; break;
case 2 s="март"; break;
case 3 s="апрель"; break;
case 4 s="май"; break;
case 5 s="июнь"; break;
case 6 s="июль"; break;
case 7 s="август"; break;
case 8 s="сентябрь"; break;
case 9 s="октябрь"; break;
case 10: s="ноябрь"; break;
case 11: s="декабрь"; break;
}
return s
```

```
}
//-->
</script>
</HEAD>
<BODY bgcolor="#FFFFCC">
<H4>В какие месяцы заданного года число 13 попадает на пятницу?</h4>
<FORM name="form1">
Введите год: <input type="text" size=8 name="fye" >
<input type="button" value=Найти onClick="def13 (form1)"><br>
<textarea Cols=30 rows=4 name=res></textarea><br>
<input type="reset" value=Отменить>
</FORM>
</BODY>
</HTML>
```

Задания

1. Проверьте примеры из лабораторной работы.
2. Напишите сценарий, который по заданной дате определяет номер недели в году.
3. Напишите сценарий, который по дате рождения человека определяет, под каким знаком зодиака родился человек.
4. В старояпонском календаре был принят 60-летний цикл, состоящий из пяти 12-летних подциклов. Подциклы обозначались названиями цвета: зеленый, красный, желтый, белый, черный. Внутри каждого подцикла годы носили названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, курицы, собаки и свиньи. Например, 1984 год (год зеленой крысы) был началом очередного цикла. Напишите сценарий, который по заданной дате определяет название года по старояпонскому календарю.

Лабораторная работа №11
Работа со строками

Строковые литералы или строковые переменные являются в языке JavaScript объектом типа string, к которому могут быть применены методы, определенные в языке. Создание нового объекта требует вызова функции-конструктора объекта. Для того чтобы создать строковый объект, надо применить конструктор newstring, например:

s=newString("результат=")

Объект string имеет единственное свойство length (длина_строки). Выражение s.length выдает значение 10, равное длине строки, содержащейся в строковом объекте s. Объект string имеет два типа методов. С методами, непосредственно влияющими на саму строку, мы сейчас и познакомимся, рассматривая примеры обработки текстовой информации.

Одним из часто используемых методов является метод выделения из строки отдельного символа. Метод charAt(ni) возвращает символ, позицию которого определяет параметр ni. Символы в строке перенумерованы, начиная с 0.

Пример 1. Вывод символов строки в "столбик"

Напишем сценарий, при выполнении которого заданный текст выводится в "столбик", т.е. на каждой строке размещается по одному символу.

При решении задачи из заданной строки последовательно выбираются символы. Формируется новая строка, в которой за каждым символом ставится последовательность символов, обеспечивающая переход на новую строку. Когда строка результата сформирована, то она размещается в текстовом поле формы, тем самым для исходной строки осуществляется вывод в "столбик". Сценарий, осуществляющий обработку строки, приведен в листинге 1.

Листинг 1. Вывод символов строки в "столбик"

```
<HTML>
<HEAD>
<TITLE>Вывод символов строки в "столбик"</TITLE>
<script language="JavaScript">
<!-- //
function ttest(s)
{ var sres="Прочитанный текст:"+" \r\n"+s+"\r\n"+
'Текст в "столбик":'+"\r\n"
var cur=""
for ( var i=0; i <= s.length-1; i += 1)
```

```
{c=s.charAt(i); cur +=c+"\r\n" }
sres+=cur
return sres
}
//-->
</script>
</HEAD>
<BODY bgcolor="#FFFFCC">
<H4>Символы текущей строки в столбик</H4>
<FORM name="form1">
Введите строку: <input type="text" size=20 name="st1"><hr>
<input type="button" value=Выполнить
onClick="form1.res.value=ttest(form1.st1.value)">
<input type="reset" value=Очистить><hr>
<textarea cols=20 rows=7 name= res></textarea>
</FORM>
</BODY>
</HTML>
```

Метод substr (n1,n2) позволяет выделять из строки подстроку. Параметр n1 задает позицию первого символа подстроки; параметр n2 определяет количество символов в подстроке. Например, если строка s="сборник", то в результате выполнения substr (0,4) будет выделена подстрока "сбор".

Пример 2. Вычисление количества повторений строки в тексте

Напишем программу, которая определяет, сколько раз заданное слово встречается в определенном тексте.

В тексте слова разделяются пробелами. После того как очередное слово найдено, просмотр продолжается с символа, следующего за найденным словом.

HTML-код документа представлен в листинге 2.

Листинг 2. Количество заданных слов в тексте

```
<HTML>
<HEAD>
<TITLE>Количество заданных слов в тексте</TITLE>
<script language="JavaScript">
<!-- //
function numword(obj)
{ var h=obj.data.value
var s=obj.textin.value
s=' '+s+' '
```

```
h=' '+h+' '
var m=h.length
var res=0
var i=0
while (i <= s.length-1)
{ ch=s.substr(i,m)
if (ch==h) {res+=l; i = i+m-1}
else
i++
}
obj.result.value=res
}
//-->
</script>
</HEAD>
<BODY bgcolor="#FFFFCC">
<H4>Количество заданных слов в тексте</H4>
<FORM name="form1">
Введите текст:<br>
<textarea name="textin" rows=4 cols=20></textarea><hr>
Введите слово: <input type="text" name="data" size="8"><hr>
<input type="button" value="0пределить" onClick="numword(form1)"><hr>
Количество слов в тексте: <input type="text" name="result" size=8><hr>
<input type="reset" value="Отменить">
</FORM>
</BODY>
</HTML>
```

Упражнения

1. Проверить примеры из лабораторной работы.
2. Слова в заданном тексте разделяются пробелами. Напишите программу, которая определяет количество слов в тексте.
3. Напишите программу, в которой все слова A заменены словом B, где A и B - заданные слова, возможно, различной длины.
4. Напишите программу, которая "сжимает" заданный текст, т. е. заменяет все подряд идущие пробелы на один.

Лабораторная работа №12
Массивы

Тип Array введен в JavaScript для возможности манипулирования самыми разными объектами, которые может отображать Navigator. Это - список всех гипертекстовых ссылок данной страницы, список всех картинок на данной странице, список всех элементов формы и т.п. Пользователь может создать и свой собственный массив, используя, конструктор Array(). Делается это следующим образом:

```
new_array = new Array()
new_array5 = new Array(5)
colors = new Array("red", "white", "blue")
```

Размерность массива может изменяться. Можно сначала определить массив, а потом присвоить одному из его элементов значение. Как только это произойдет, изменится и размерность массива:

```
colors = new Array()
colors[5] = "red".
```

В данном случае массив будет состоять из 6 элементов, так как первым элементом массива считается элемент с индексом 0.

Для массивов определены четыре метода: join, reverse, sort, concat. Join объединяет элементы массива в строку символов, в качестве аргумента в этом методе задается разделитель:

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

В результате выполнения присваивания значения строке символов string мы получим следующую строку: string = "red + white + blue". Другой метод, reverse, изменяет порядок элементов массива на обратный, метод sort отсортировывает их в лексикографическом порядке, а метод concat объединяет два массива.

У массивов есть два свойства: length и prototype. Length определяет число элементов массива. Если нужно выполнить некоторую рутинную операцию над всеми элементами массива, то можно воспользоваться циклом типа:

```
color = new Array("red", "white", "blue")
n = 0 while(n != colors.length)
{... операторы тела цикла ...}
```

Свойство prototype позволяет добавить свойства к объектам массива. Однако чаще всего в программах на JavaScript используются встроенные массивы, в основном графические образы (Images) и гипертекстовые ссылки (Links).

В новой версии языка появился конструктор для этого типа объектов:

```
new_image = new Image()
 new_image = new Image(width, height)
```

Пример 1. Создание мультипликации с использованием массивов.

Часто для создания мультипликации формируют массив графических объектов, которые потом прокручивают один за другим:

```
img_array = new Array()
 img_array[0] = new Image(50,100)
 img_array[1] =new Image(50,100)
  ...
 img_array[99] = new Image(50,100)
```

У объекта Image существует 10 свойств, из которых, пожалуй, самым важным является src. Так, для присваивания конкретных картинок элементам массива img_array следует воспользоваться следующей последовательностью команд:

```
img_array[0].src = "image1.gif"
 img_array[1].src = "image2.gif"
 ...
 img_array[99].src = "image100.gif"
```

В данном случае можно было воспользоваться и циклом для присвоения имен, так как они могут быть составлены из констант и значения индексной переменной.

Расширяя пример с массивом Image, построим теперь документ, в котором будет встроена мультипликация, определенная нашим массивом:

Листинг 1. Мультипликация.

```
 <HTML>
 <HEAD>
 <SCRIPT>
 <!--//
 function multi_pulti()
 {
 img_array = new Array()
 for (var i=0; i<4; i++)
 img_array[i]  = new Image(50,100)
 img_array[0].src = "e1.jpg"
 img_array[1].src = "e2.jpg"
 img_array[2].src = "e3.jpg"
 img_array[3].src = "e4.jpg"
 var t=new Date()
 p=-1
```

```
}

function s()
{
p=p+1
document.images[0].src =img_array[p].src
setTimeout("s()",100)
if (p==3) p=-1
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad="multi_pulti()">
<img src="e1.jpg">
<br>
<input type="Button" name="But" value="Посмотреть" onClick="s()">
</BODY>
</HTML>
```

Далее рассмотрим несколько классических задач, посвященных работе с массивами. Приведем функции работы с массивами, которые ценны сами по себе и могут применяться в различных программах.

Пример 2. Бинарный поиск с формированием таблицы результатов

Напишем функцию, которая реализует алгоритм бинарного поиска таким образом, чтобы во время работы программы формировалась таблица значений переменных i, j, k и некоторых выражений.

Листинг 2. Поиск в упорядоченном массиве с таблицей промежуточных значений

```
<HTML>
<HEAD>
<TITLE>Бинарный поиск. Таблица промежуточных значений</TITLE>
<script language="JavaScript">
<!-- //
var v=new Array (2, 3, 5, 6, 6, 7,10,11, 20, 25)
function testtab(obj,v,t)
{ var res="i j k v[k] t<= v[k]"+"\r\n"
var i=0
var j= v.length-1
var k
while ( i < j )
```

```
{ k=Math.round( (i+j)/2+0.5)-1
res = res + i + " "+j+" "+k+" "+"v[" + k + "]=" + v[k] + " " + t + "<=" + v[k]+"\r\n"
if (t <= v[k] )
j=k
else
i=k+l
}
res += "v[" + i + "]=" +v[i]+"\r\n"
obj.resultl.value=res
if (v[i] == t )
{ return i}
else return -1
}
function test(obj)
{ obj.datal.yalue=v}
//-->
</script>
</HEAD>
<BODY bgcolor=silver>
<H4>Реализация алгоритма бинарного поиска</H4>
<FORM name="forml">
<pre>
Массив:<INPUT type="text" size=40 name="datal" ><hr>
Элемент:<INPUT type="text" size=20 name="data2" ><hr>
Результат поиска: <INPUT type="text" size=20 name="result" ><hr>
Таблица промежуточных значений: <BR>
<textarea cols=50 rows=7 name="result1" > </textarea><hr>
</PRE>
<input type="button" value=0пределить onClick="test(form1); forml.result.value=testtab(form1,v,forml.data2.value)">
<input type="reset" value=Отменить>
</FORM>
</BODY>
</HTML>
```

Задания

1. Проверить пример 2 из лабораторной работы.
2. Создать простейший мультипликационный сюжет с использованием массива.
3. Задан одномерный массив вещественных чисел. Напишите сценарий, который определяет число положительных элементов массива.
4. Задан одномерный массив вещественных чисел. Напишите сценарий, позволяющий найти максимальный элемент в массиве.

Лабораторная работа №13
Формы

Цели: Разобрать способы выполнения проверки формы, а также возможности для пересылки информации по Интернет.

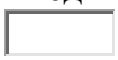
Проверка информации, введенной в форму

Формы широко используются на Интернет. Информация, введенная в форму, часто посылается обратно на сервер или отправляется по электронной почте на некоторый адрес. Проблема состоит в том, чтобы убедиться, что введенная пользователем в форму информация корректна. Легко проверить ее перед пересылкой в Интернет можно с помощью языка JavaScript.

Сперва нам необходимо создать простой скрипт. Допустим, HTML-страница содержит два элемента для ввода текста. В первый из них пользователь должен вписать свое имя, во второй элемент - адрес для электронной почты. Вы можете ввести туда какую-нибудь информацию и нажать клавишу. Попробуйте также нажать клавишу, не введя в форму никакой информации.

Введите Ваше имя:

Введите Ваш адрес e-mail:



Что касается информации, введенной в первый элемент, то Вы будете получать сообщение об ошибке, если туда ничего не было введено. Любая представленная в элементе информация будет рассматриваться на предмет корректности. Конечно, это не гарантирует, что пользователь введет не то имя. Браузер даже не будет возражать против чисел. Например, если Вы введете '17', то получите приглашение 'Hi 17!'. Так что эта проверка не может быть идеальна.

Второй элемент формы несколько более сложнее. Попробуйте ввести простую строку - например Ваше имя. Сделать это не удастся до тех пор, пока Вы не укажите @ в Вашем имени... Признаком того, что пользователь правильно ввел адрес электронной почты служит наличие символа @. Этому условию будет отвечать и одиночный символ @, даже несмотря на то, что это бессмысленно. В Интернет каждый адрес электронной почты содержит символ @, так что проверка на этот символ здесь уместна.

Как скрипт работает с этими двумя элементами формы и как выглядит проверка? Это происходит следующим образом:

```
<html>
<head>
<script language="JavaScript">
<!-- Скрыть

function test1(form) {
  if (form.text1.value == "")
    alert("Пожалуйста, введите строку!")
```

```
  else {
   alert("Hi "+form.text1.value+"! Форма заполнена корректно!");
  }
}
function test2(form) {
  if (form.text2.value == "" ||
      form.text2.value.indexOf('@', 0) == -1)
       alert("Неверно введен адрес e-mail!");
  else alert("OK!");
}
// -->
</script>
</head>
<body>
<form name="first">
Введите Ваше имя:<br>
<input type="text" name="text1">
<input type="button" name="button1" value="Проверка"
onClick="test1(this.form)">
<P>
Введите Ваш адрес e-mail:<br>
<input type="text" name="text2">
<input type="button" name="button2" value="Проверка"
nClick="test2(this.form)">
</body>
</html>
```

Рассмотрим сначала HTML-код в разделе body. Здесь мы создаем лишь два элемента для ввода текста и две кнопки. Кнопки вызывают функции test1(...) или test2(...), в зависимости от того, которая из них была нажата. В качестве аргумента к этим функциям мы передаем комбинацию *this.form*, что позже позволит нам адресоваться в самой функции именно к тем элементам, которые нам нужны.

Функция test1(form) проверяет, является ли данная строка пустой. Это делается посредством *if (form.text1.value == "")...* . Здесь 'form' - это переменная, куда заносится значение, полученное при вызове функции от 'this.form'. Мы можем извлечь строку, введенную в рассматриваемый элемент, если к *form.text1* припишем 'value'. Чтобы убедиться, что строка не является пустой, мы сравниваем ее с "". Если же окажется, что введенная строка соответствует "", то это значит, что на самом деле ничего введено не было. И наш пользователь получит сообщение об ошибке. Если же что-то было введено верно, пользователь получит подтверждение - ok.

Следующая проблема заключается в том, что пользователь может вписать в поле формы одни пробелы. И это будет принято, как корректно введенная информация! Если есть желание, то Вы конечно можете добавить проверку такой возможности и исключить ее.

Рассмотрим теперь функцию *test2(form)*. Здесь вновь сравнивается введенная строка с пустой - "" (чтобы удостовериться, что что-то действительно было введено читателем). Однако к команде if мы добавили еще кое-чего. Комбинация символов || называется оператором OR (ИЛИ). Команда if проверяет, чем заканчивается первое или второе сравнения. Если хотя бы одно из них выполняется, то и в целом команда if имеет результатом true, а стало быть будет выполняться следующая команда скрипта. Словом, Вы получите сообщение об ошибке, если либо предоставленная Вами строка пуста, либо в ней отсутствует символ @. (Второй оператор в команде if следит за тем, чтобы введенная строка содержала @.)

Проверка на присутствие определенных символов

В некоторых случаях Вам понадобится ограничивать информацию, вводимую в форму, лишь некоторым набором символов или чисел. Достаточно вспомнить о телефонных номерах - представленная информация должна содержать лишь цифры (предполагается, что номер телефона, как таковой, не содержит никаких символов). Нам необходимо проверять, являются ли введенные данные числом. Сложность ситуации состоит в том, что большинство людей вставляют в номер телефона еще и разные символы - например: 01234-56789, 01234/56789 or 01234 56789 (с символом пробела внутри). Не следует принуждать пользователя отказываться от таких символов в телефонном номере. А потому мы должны дополнить наш скрипт процедурой проверки цифр и некоторых символов.

Telephone:

Исходный код этого скрипта:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function check(input) {
  var ok = true;
  for (var i = 0; i < input.length; i++) {
    var chr = input.charAt(i);
    var found = false;
    for (var j = 1; j < check.length; j++) {
      if (chr == check[j]) found = true;
    }
    if (!found) ok = false;
  }
   return ok;
}
function test(input) {

  if (!check(input, "1", "2", "3", "4",
       "5", "6", "7", "8", "9", "0", "/", "-", " ")) {
```

```
    alert("Input not ok.");
   }
   else {
    alert("Input ok!");
   }
  }
  // -->
  </script>
  </head>
  <body>
  <form>
  Telephone:
  <input type="text" name="telephone" value="">
  <input type="button" value="Check"
   onClick="test(this.form.telephone.value)">
  </form>
  </body>
  </html>
```

Функция test() определяет, какие из введенных символов признаются корректными.

Предоставление информации, введенной в форму

Какие существуют возможности для передачи информации, внесенной в форму? Самый простой способ состоит в передаче данных формы по электронной почте (этот метод мы рассмотрим поподробнее). Если Вы хотите, чтобы за вносимыми в форму данными следил сервер, то Вы должны использовать интерфейс CGI (Common Gateway Interface). Последнее позволяет Вам автоматически обрабатывать данные. Например, сервер мог бы создавать базу данных со сведениями, доступную для некоторых из клиентов. Другой пример - поисковые страницы, такие как Yahoo. Обычно в них представлена форма, позволяющая создавать запрос для поиска в собственной базе данных. В результате пользователь получает ответ вскоре после того, как нажимает на соответствующую кнопку. Ему не приходится ждать, пока люди, отвечающие за поддержание данного сервера, прочтут указанные им данные и отыщут требуемую информацию. Все это автоматически выполняет сам сервер. JavaScript не позволяет делать таких вещей.

С помощью JavaScript Вы не сможете создать книгу читательских отзывов, поскольку JavaScript лишен возможности записывать данные в какой-либо файл на сервере. Делать это Вы можете только через интерфейс CGI. Конечно, Вы можете создать книгу отзывов, для которой пользователи присылали сведения по электронной почте. Однако в этом случае Вы должны заносить отзывы вручную. Так можно делать, если Вы не предполагаете получать ежедневно по 1000 отзывов.

Соответствующий скрипт будет простым текстом HTML.

<form method=post action=mailto:your.address@goes.here

```
enctype="text/plain">
Нравится ли Вам эта страница?
  <input name="choice" type="radio" value="1">Вовсе нет.<br>
  <input name="choice" type="radio" value="2" CHECKED>Напрасная трата
времени.<br>
  <input name="choice" type="radio" value="3">Самый плохой сайт в
Сети.<br>
  <input name="submit" type="submit" value="Send">
</form>
```

Параметр *enctype="text/plain"* используется для того, чтобы пересылать именно простой текст без каких-либо кодируемых частей.

Если Вы хотите проверить форму прежде, чем она будет передана в сеть, то для этого можете воспользоваться программой обработки событий onSubmit. Вы должны поместить вызов этой программы в тэг <form>. Например:

```
function validate() {
  // check if input ok
  // ...
  if (inputOK) return true
    else return false;
}
...
<form ... onSubmit="return validate()">
...
```

Форма, составленная таким образом, не будет послана в Интернет, если в нее внесены некорректные данные.

Выделение определенного элемента формы

С помощью метода focus() Вы можете сделать вашу форму более дружественной. Так, Вы можете выбрать, который элемент будет выделен в первую очередь. Либо Вы можете приказать браузеру выделить ту форму, куда были введены неверные данные. Сделать это Вы можете с помощью следующего фрагмента скрипта:

```
function setfocus() {
  document.first.text1.focus();
}
```

Эта запись могла бы выделить первый элемент для ввода текста в скрипте. Вы должны указать имя для всей формы - в данном случае она называется *first* - и имя одного элемента формы - *text1*. Если Вы хотите, чтобы при загрузке страницы данный элемент выделялся, то для этого Вы можете дополнить тэг <body> атрибутом onLoad. Это будет выглядеть как:

```
<body onLoad="setfocus()">
```

Остается еще дополнить пример следующим образом:

```
function setfocus() {
  document.first.text1.focus();
  document.first.text1.select();
}
```

Попробуйте следующий код:

bla bla bl

При этом не только будет выделен элемент, но и находящийся в нем текст.

Контрольные вопросы:

1. Для чего иногда нужно ограничивать информацию, вводимую в форму? Как это сделать на JavaScript?
2. Какие существуют способы передачи данных формы?
3. Для чего предназначен метод focus()?
4. В каких целях используется программа обработки событий onSubmit?

Лабораторная работа № 14
Объект Image

Цели: Изучить возможности объекта Image. Разобрать способы загрузки изображений.

Изображения на web-странице

Рассмотрим теперь объект Image, который стал доступен, начиная с версии с 1.1 языка JavaScript (то есть с Netscape Navigator 3.0). С помощью объекта Image Вы можете вносить изменения в графические образы, присутствующие на web-странице. В частности, это позволяет нам создавать мультипликацию.

Заметим, что пользователи браузеров более старых версий (таких как Netscape Navigator 2.0 или Microsoft Internet Explorer 3.0 - т.е. использующих версию 1.0 языка JavaScript) не смогут запускать скрипты, приведенные в этой части описания. Или, в лучшем случае, на них нельзя будет получить польный эффект.

Давайте сначала рассмотрим, как из JavaScript можно адресоваться к изображениям, представленным на web-странице. В рассматриваемом языке все изображения предстают в виде массива. Массив этот называется *images* и является свойством объекта document. Каждое изображение на web-странице получает порядковый номер: первое изображение получает номер 0, второе - номер 1 и т.д. Таким образом, к первому изображению мы можем адресоваться, записав document.images[0].

Каждое изображение в HTML-документе рассматривается в качестве объекта Image. Объект Image имеет определенные свойства, к которым и можно обращаться из языка JavaScript. Например, Вы можете определить, который размер имеет изображение, обратившись к его свойствам *width* и

height. То есть по записи document.images[0].width Вы можете определить ширину первого изображения на web-странице (в пикселях).

К сожалению, отслеживать индекс всех изображений может оказаться затруднительным, особенно если на одной странице у Вас их довольно много. Эта проблема решается назначением изображениям своих собственных имен. Так, если Вы заводите изображение с помощью тэга

<img src="img.gif" name="myImage" width=100 height=100>

то Вы сможете обращаться к нему, написав document.myImage или document.images["myImage"].

Загрузка новых изображений

Хотя конечно и хорошо знать, как можно получить размер изображения на web-странице, это не совсем то, чего бы мы хотели. Мы хотим осуществлять смену изображений на web-странице и для этого нам понадобится атрибут *src*. Как и в случае тэга <img>, атрибут *src* содержит адрес представленного изображения. Теперь - в языке JavaScript 1.1 - Вы имеете возможность назначать новый адрес изображению, уже загруженному в web-страницу. И в результате, изображение будет загружено с этого нового адреса, заменив на web-странице старое. Рассмотрим к примеру запись:

<img src="img1.gif" name="myImage" width=100 height=100>

Здесь загружается изображение *img1.gif* и получает имя *myImage*. В следующей строке прежнее изображение *img1.gif* заменяется уже на новое - *img2.gif*:

document.myImage.src= "img2.src";

При этом новое изображение всегда получает тот же размер, что был у старого. И Вы уже не можете изменить размер поля, в котором это изображение размещается.

Упреждающая загрузка изображения

Один из недостатков такого подхода может заключаться в том, что *после* записи в *src* нового адреса начинает процесс загрузки соответствующего изображения. И поскольку этого не было сделано заранее, то еще пройдет некоторое время, прежде чем новое изображение будет передано через Интернет и встанет на свое место. В некоторых ситуациях это допустимо, однако часто подобные задержки неприемлемы. И что же мы можем сделать с этим? Конечно, решением проблемы была бы упреждающая загрузка изображения. Для этого мы должны создать новый объект Image. Рассмотрим следующие строки:

hiddenImg= new Image();

hiddenImg.src= "img3.gif";

В первой строке создается новый объект Image. Во второй строке указывается адрес изображения, которое в дальнейшем будет представлено с помощью объекта *hiddenImg*. Как мы уже видели, запись нового адреса в атрибуте *src* заставляет браузер загружать изображение с указанного адреса.

Поэтому, когда выполняется вторая строка нашего примера, начинает загружаться изображение *img2.gif*. Но как подразумевается самим названием *hiddenImg* ("скрытая картинка"), после того, как браузер закончит загрузку, изображение на экране не появится. Оно будет лишь будет сохранено в памяти компьютера (или точнее в кэше) для последующего использования. Чтобы вызвать изображение на экран, мы можем воспользоваться строкой:

document.myImage.src= hiddenImg.src;

Но теперь изображение уже немедленно извлекается из кэша и показывается на экране. Таким образом, сейчас мы управляли упреждающей загрузкой изображения.

Конечно браузер должен был к моменту запроса закончить упреждающую загрузку, чтобы необходимое изображение было показано без задержки. Поэтому, если Вы должны предварительно загрузить большое количество изображений, то может иметь место задержка, поскольку браузер будет занят загрузкой всех картинок. Вы всегда должны учитывать скорость связи с Интернет - загрузка изображений не станет быстрее, если пользоваться только что показанными командами. Мы лишь пытаемся чуть раньше загрузить изображение - поэтому и пользователь может увидеть их раньше. В результате и весь процесс пройдет более гладко.

Изменение изображений в связи с событиями

Вы можете создать красивые эффекты, используя смену изображений в качестве реакции на определенные события. Например, Вы можете изменять изображения в тот момент, когда курсор мыши попадает на определенную часть страницы. Проверьте, как работает следующий пример, просто поместив курсор мыши на картинку (впрочем, при этом Вы получите сообщение об ошибке, если пользуетесь браузером, поддерживающим лишь JavaScript 1.)

Исходный код этого примера выглядит следующим образом:

```
<a href="#"
  onMouseOver="document.myImage2.src='img2.gif'"
  onMouseOut="document.myImage2.src='img1.gif'">
<img src="img1.gif" name="myImage2" width=160 height=50 border=0></a>
```

При этом могут возникнуть следующие проблемы:

- пользователь пользуется браузером, не имеющим поддержки JavaScript 1.1.
- Второе изображение не было загружено.
- Для этого мы должны писать новые команды для каждого изображения на web-странице.

- Мы хотели бы иметь такой скрипт, который можно было бы использовать во многих web-страницах вновь и вновь, и без больших переделок.

Теперь мы рассмотрим полный вариант скрипта, который мог бы решить эти проблемы. Хотя скрипт и стал намного длиннее - но написав его один раз, Вы не больше будете беспокоиться об этих проблемах.

Чтобы этот скрипт сохранял свою гибкость, следует соблюдать два условия:

- Не оговоривается количество изображений - не должно иметь значения, сколько их используется, 10 или 100
- Не оговоривается порядок следования изображений - должна существовать возможность изменять этот порядок без изменения самого кода

Посмотрим этот код в работе:

Рассмотрим скрипт:

```
<html>
<head>

<script language="JavaScript">
<!-- hide

  // ok, у нас браузер с поддержкой JavaScript
  var browserOK = false;
  var pics;

// -->
</script>

<script language="JavaScript1.1">
<!-- hide

  // браузер с поддержкой JavaScript 1.1!
  browserOK = true;
  pics = new Array();

// -->
</script>



<script language="JavaScript">
```

```
<!-- hide

var objCount = 0; // количество изображений на  web-странице

function preload(name, first, second) {

  // предварительна\я загрузка изображений и размещение их в массиве

  if (browserOK) {
    pics[objCount] = new Array(3);
    pics[objCount][0] = new Image();
    pics[objCount][0].src = first;
    pics[objCount][1] = new Image();
    pics[objCount][1].src = second;
    pics[objCount][2] = name;
    objCount++;
  }
}

function on(name){
  if (browserOK) {
     for (i = 0; i < objCount; i++) {
      if (document.images[pics[i][2]] != null)
        if (name != pics[i][2]) {
          // вернуть в исходное систо\яние все другие изображени\я
          document.images[pics[i][2]].src = pics[i][0].src;
        } else {
          // показывать вторую картинку, поскольку курсор пересекает данное изображение
          document.images[pics[i][2]].src = pics[i][1].src;
        }
    }
  }
}

function off(){
  if (browserOK) {
     for (i = 0; i < objCount; i++) {
      // вернуть в исходное систо\яние все изображени\я
      if (document.images[pics[i][2]] != null)
        document.images[pics[i][2]].src = pics[i][0].src;
    }
  }
}
```

```
// заранее загружаемые изображения - Вы должны здесь указать
// изображения, которые нужно загрузить заранее, а также объект Image,
// к которому они относятся (первый аргумент). Именно эту часть
// нужно корректировать, если Вы хотите использовать скрипт
// применительно к другим изображениям (конечно это не освобождает
// Вас от обязанности подредактировать в документе также и раздел body)

preload("link1", "img1f.gif", "img1t.gif");
preload("link2", "img2f.gif", "img2t.gif");
preload("link3", "img3f.gif", "img3t.gif");

// -->
</script>
<head>



<body>
<a href="link1.htm" onMouseOver="on('link1')"
  onMouseOut="off()">
<img name="link1" src="link1f.gif"
  width="140" height="50" border="0"></a>

<a href="link2.htm" onMouseOver="on('link2')"
  onMouseOut="off()">
<img name="link2" src="link2f.gif"
  width="140" height="50" border="0"></a>

<a href="link3.htm" onMouseOver="on('link3')"
  onMouseOut="off()">
<img name="link3" src="link3f.gif"
  width="140" height="50" border="0"></a>
</body>
</html>
```

Данный скрипт помещает все изображения в массив *pics*. Создает этот массив функция preload(), которая вызвается в самом начале. Вызов функции preload() выглядит просто как:

```
preload("link1", "img1f.gif", "img1t.gif");
```

Это означает, что скрипт должен загрузить с сервера два изображения: *img1f.gif* и *img1t.gif*. Первое из них - это та картинка, которая будет

представлена, пока курсор мыши не попадает в область изображение. Когда же пользователь помещает курсор мыши на изображение, то появляется вторая картинка. При вызове функции preload() в качестве первого аргумента мы указываем слово *"link1"* и тем самым задаем на web-странице объект Image, которому и будут принадлежать оба предварительно загруженных изображения. Если Вы посмотрите в нашем примере в раздел <body>, то обнаружите изображение с тем же именем *link1*. Мы пользуем не порядковый номер, а именно имя изображения для того, чтобы иметь возможность переставлять изображения на web-странице, не переписывая при этом сам скрипт.

Обе функции on() и off() вызываются посредством программ обработки событий onMouseOver и onMouseOut. Поскольку сам элемент image не может отслеживать события MouseOver и MouseOut, то мы обязаны сделать на этих изображениях еще и ссылки.

Можно видеть, что функция on() возвращает все изображения, кроме указанного, в исходное состояние. Делать это необходимо потому, что в противном случае выделеными могут оказаться сразу несколько изображений (дело в том, что событие MouseOut не будет зарегистрировано, если пользователь переместит курсор с изображения сразу за пределы окна).

Изображения - без сомнения могучее средство улучшения Вашей web-страницы. Объект Image дает Вам возможность создавать действительно сложные эффекты. Однако заметим, что *не всякое* изображение или программа JavaScript способно улучшить Вашу страницу. Если Вы пройдетесь по Сети, то сможете увидеть множество примеров, где изображения использованы самым ужасным способом. Не количество изображений делает Вашу web-страницу привлекательной, а их качество. Сама загрузка 50 килобайт плохой графики способна вызвать раздражение. При создании специальных эффектов с изображениями с помощью JavaScript помните об этом и ваши посетителями/клиентами будут чаще возвращаться на Ваши страницы.

Контрольные вопросы:

1. Для чего предназначен объект Image?
2. Как из JavaScript можно адресоваться к изображениям?
3. Для чего служит атрибут *src* тэга <img>?
4. Чтобы скрипт смены изображений сохранял свою гибкость, какие условия следует соблюдать?
5. Объясните назначение функции функция preload().