МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»
(КНИТУ-КАИ)
Чистопольский филиал «Восток»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

ТЕХНОЛОГИИ РАЗРАБОТКИ ВЕБ-СИСТЕМ

Индекс по учебному плану: **Б1.В.10**
Направление подготовки: **09.03.01 Информатика и вычислительная техника**
Квалификация: **Бакалавр**
Профиль подготовки: **Автоматизированные системы обработки информации и управления**
Типы задач профессиональной деятельности: **проектный, производственно-технологический**

Рекомендовано УМК ЧФ КНИТУ-КАИ

Чистополь
2023 г.

Лабораторное занятие № 1

Тема: Работа с JavaScript. Размещение JavaScript на HTML странице
Цель занятия: Ознакомить с основами языка JavaScript
Задания:

1 Ознакомьтесь с теоретическими аспектами темы.

2 Создайте простую веб-страницу с использованием JavaScript согласно методическим указаниям.

3 Создайте веб-страницу с формой и кнопкой на основе JavaScript согласно методическим указаниям.

4 Напишите скрипт, печатающий текст «Добро пожаловать на мою страницу! Это JavaScript» три раза подряд. Инструкция по выполнению задания представлена в методических указаниях.

5 Создайте веб-страницу с использованием функции calculation().

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

JavaScript - новый язык для составления скриптов, разработанный фирмой Netscape. С помощью JavaScript Вы можете легко создавать интерактивные Web-страницы.

Для запуска скриптов, написанных на языке JavaScript, нужен браузер, способный работать с JavaScript - например Netscape Navigator (начиная с версии 2.0) или Microsoft Internet Explorer (MSIE - начиная с версии 3.0). С тех пор, как оба этих браузера стали широко распространеными, множество людей получили возможность работать со скриптами, написанными на языке JavaScript. Код скрипта JavaScript размещется непосредственно на HTML-странице. Все, что стоит между тэгами <script> и </script>, интерпретируется как код на языке JavaScript. Инструкция*document.write()* - одна из наиболее важных команд, используемых при программировании на языке JavaScript. Команда *document.write()* используется, когда необходимо что-либо написать в текущем документе (в данном случае таком является наш HTML-документ).

События и обработчики событий являются очень важной частью для программирования на языке JavaScript. События, главным образом, инициируются теми или иными действиями пользователя. Если он щелкает по некоторой кнопке, происходит событие *"Click"*. Если указатель мыши пересекает какую-либо ссылку гипертекста - происходит событие *MouseOver*. Существует несколько различных типов событий. Мы можем заставить нашу JavaScript-программу реагировать на некоторые из них. И это может быть выполнено с помощью специальных программ обработки событий. Так, в результате щелчка по кнопке может создаваться выпадающее окно. Это означает, что создание окна должно быть реакцией на событие щелка - *Click*. Программа - обработчик событий, которую мы должны использовать в данном

случае, называется *onClick*. И она сообщает компьютеру, что нужно делать, если произойдет данное событие.

Вы можете использовать в скрипте множество различных типов функций обработки событий. В большинстве случаев функции представляют собой лишь способ связать вместе нескольких команд. Функции могут также использоваться в совместно с процедурами обработки событий.

Методические рекомендации к выполнению задания 2

1. Запустите блокнот
2. Введите текст

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
<scriptlanguage="JavaScript">
   document.write("Аэто JavaScript!")
</script>
<br>
Вновь документ HTML.
</body>
</html>
```

3. Сохраните документ в формате html
4. Запустите страницу в окне браузера.

Результат выполнения файла в случае, если используемый браузер поддерживает JavaScript:

```
Это      обычный      HTML
документ.
А это JavaScript!
```

Если браузер не поддерживает JavaScript, то он проигнорирует тег <script>. В этом случае измените исходный текст:

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
<script language="JavaScript">
<!-- hide from old browsers
   document.write("Аэто JavaScript!")
// -->
</script><br>
Вновь документ HTML.
</body>
</html>
```

В этом случае использован тег комментария из HTML - <!-- -->. В результате новый вариант нашего исходного кода будет выглядеть как:

Это обычный HTML документ.

Вновь документ HTML.

Методические рекомендации к выполнению задания 3

1. Создайте новый документ html.
2. Вставьте следующий код

```
<form>
<input type="button" value="Click me" onClick = "alert('Yo')">
</form>
```

3. Просмотрите результат.

В данном примере создается некая форма с кнопкой. Первая новая особенность - onClick="alert('Yo')" в тэге <input>. Таким образом, если имеет место событие *Click*, компьютер должен выполнить вызов alert('Yo'). Это и есть пример кода на языке JavaScript.

Функция *alert()* позволяет Вам создавать выпадающие окна. При ее вызове Вы должны в скобках задать некую строку. В нашем случае это *'Yo'*. И это как раз будет тот текст, что появится в выпадающем окне. Таким образом, когда читатель когда щелкает на кнопке, наш скрипт создает окно, содержащее текст *'Yo'*.

Еще одна особенность данного примера: в команде document.write() мы использовали двойные кавычки ("), а в конструкции alert() - только одинарные. В большинстве случаев Вы можете использовать оба типа кавычек. Однако в последнем примере мы написали onClick="alert('Yo')" - то есть мы использовали и двойные, и одинарные кавычки. Если бы мы написали onClick="alert("Yo")", то компьютер не смог бы разобраться в нашем скрипте, поскольку становится неясно, к которой из частей конструкции имеет отношение функция обработки событий onClick, а к которой - нет. Поэтому мы вынуждены использовать оба типа кавычек. Не имеет значения, в каком порядке Вы использовали кавычки - сначала двойные, а затем одинарные или наоборот. То есть Вы можете точно так же написать и onClick='alert("Yo")'.

Методические рекомендации к выполнению задания 4

1. Напишите скрипт, печатающий текст «Добро пожаловать на мою страницу! Это JavaScript» три раза подряд. Для начала рассмотрим простой подход:

```
<html>
<script language="JavaScript">
<!-- hide
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
document.write("Добро пожаловать на мою страницу!<br>");
```

```
document.write("Это JavaScript!<br>");
// -->
</script>
</html>
```

2. Если посмотреть на исходный код скрипта, то видно, что для получения необходимого результата определенная часть его кода была повторена три раза. Эту же задачу можно решить несколько иначе. Введите изменения:

```
<html>
<scriptlanguage="JavaScript">
<!-- hide
functionmyFunction() {
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");}
myFunction();
myFunction();
myFunction();
// -->
</script>
</html>
```

В этом скрипте мы определили некую функцию, состоящую из следующих строк:

```
function myFunction() {
  document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");}
```

Все команды скрипта, что находятся внутри фигурных скобок - {} - принадлежат функции *myFunction()*. Это означает, что обе команды document.write() теперь связаны воедино и могут быть выполнены при вызове указанной функции. И действительно, в нашем примере есть три вызова этой функции В свою очередь, это означает, что содержимое этой функции (команды, указанные в фигурных скобках) было выполнено трижды.

Методические рекомендации к выполнению задания 5

Создайте новую веб-страничку:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function calculation() {
  var x= 12;
  var y= 5;
  var result= x + y;
  alert(result);}
// -->
</script>
```

```
</head>
<body>
<form>
<input type="button" value="Calculate" onClick="calculation()">
</form>
</body>
</html>
```

Здесь при нажатии на кнопку осуществляется вызов функции *calculation()*. Как можно заметить, эта функция выполняет некие вычисления, пользуясь переменными *x, y* и *result*. Переменную мы можем определить с помощью ключевого слова *var*. Переменные могут использоваться для хранения различных величин - чисел, строк текста и т.д. Так строка скрипта var result= x + y; сообщает браузеру о том, что необходимо создать переменную *result* и поместить туда результат выполнения арифметической операции *x + y* (т.е. 5 + 12). После этого в переменный *result* будет размещено число *17*. В данном случае команда alert(result) выполняет то же самое, что и alert(17). Иными словами, мы получаем выпадающее окно, в котором написано число 17.

Вопросы для самоконтроля:

1 Для чего предназначен язык JavaScript?
2 Что называют инструкциями?
3 В чем отличие процедур от событий?

Лабораторное занятие № 2

Тема: Документ HTML

Цель занятия: ознакомить студентов с принципами работы с формами, иерархией объектов веб-страницы.

Задание:

1 Ознакомиться с теоретическими аспектами темы.
2 Создайте веб-страницу с использованием принципов иерархии объектов.
3 Создайте документ с использованием объектов.

Необходимые приборы: персональный компьютер, текстовый редактор Блокнот, браузер.

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

В языке JavaScript все элементы на web-странице выстраиваются в иерархическую структуру. Каждый элемент предстает в виде объекта. И каждый такой объект может иметь определенные свойства и методы. В свою очередь, язык JavaScript позволяет легко управлять объектами web-страницы, хотя для этого очень важно понимать иерархию объектов, на которые опирается разметка HTML.

С точки зрения языка JavaScript окно браузера - это некий объект window. Этот объект также содержит в свою очередь некоторые элементы оформления, такие как строка состояния. Внутри окна можно разместить документ HTML или файл какого-либо другого типа. Такая страница является объектом document. Это означает, что объект document представляет в языке JavaScript загруженный на настоящий момент документ HTML. Объект document является очень важным объектом в языке JavaScript. К свойствам объекта document относятся, например, цвет фона для web-страницы. Все без исключения объекты HTML являются свойствами объекта document. Примерами объекта HTML являются, к примеру, ссылка или заполняемая форма.

Чтобы иметь возможность получать информацию о различных объектах в этой иерархии и управлять ею мы должны знать, как в языке JavaScript организован доступ к различным объектам. Каждый объект иерархической структуры имеет свое имя. Следовательно, если нужно узнать, как можно обратиться к первому рисунку на HTML-странице, то обязаны сориентироваться в иерархии объектов. И начать нужно с самой вершины. Первый объект такой структуры называется *document*. Первый рисунок на странице представлен как объект *images[0]*. Это означает, что отныне мы можем получать доступ к этому объекту, записав в JavaScript *document.images[0]*. Если же, например, нужноузнать, какой текст ввел читатель в первый элемент формы, то нужно выяснить, как получить доступ к этому объекту. И снова начинаем с вершины нашей иерархии объектов. Затем нужно проследить путь к объекту с именем *elements[0]* и последовательно записать названия всех объектов, которые минуем. В итоге выясняется, что доступ к первому полю для ввода текста можно получить, записав: *document.forms[0].elements[0]*

Чтобы узнать текст, введенный читателем нужно написать на языке JavaScript строку:

name= document.forms[0].elements[0].value;

Полученная строка заносится в переменную *name*. Следовательно, теперь мы можем работать с этой переменной, как нам необходимо. Например, мы можем создать выпадающее окно, воспользовавшись командой alert("Hi " + name). В результате, если читатель введет в это поле слово *'Stefan'*, то по команде alert("Hi " + name) будет открыто выпадающее окно с приветствием *'Hi Stefan'*.

Если Вы имеете дело с большими страницами, то процедура адресации к различным объектам по номеру может стать весьма запутанной. Например, придется решать, как следует обратиться к объекту *document.forms[3].elements[17]document.forms[2].elements[18]*? Во избежание подобной проблемы, можно самим присваивать уникальные имена различным объектам. Посмотрим на примере, как это делается:

<formname="myForm">

Name:

<input type="text" name="name" value="">
...

Эта запись означает, что объект *forms[0]* получает теперь еще и второе имя - *myForm*. Точно так же вместо *elements[0]*можно писать *name* (последнее было указано в атрибуте name тэга <input>). Таким образом, вместо

name= document.forms[0].elements[0].value;

Вы можете записать

name= document.myForm.name.value;

Это значительно упрощает программирование на JavaScript, особенно в случае с большими web-страницами, содержащими множество объектов. При написании имен нужно следить и за положением регистра - то есть нельзя написать *myform* вместо *myForm*. В JavaScript многие свойства объектов доступны не только для чтения. Имеетсявозможность записывать в них новые значения. Например, посредством JavaScript можно записать в упоминавшееся поле новую строку.

Пример кода на JavaScript, иллюстрирующего такую возможность - интересующий нас фрагмент записан каксвойство onClick второго тэга <input>:

```
<form name="myForm">
<input type="text" name="input" value="Привет!!!">
<input type="button" value="Write"
  onClick="document.myForm.input.value= 'Yo!'; ">
```

Кроме объектов *window* и *document* в JavaScript имеется еще один важный объект - *location*. В этом объекте представлен адрес загруженного HTML-документа. Например, если Вы загрузили страницу *http://www.xyz.com/page.html*, то значение *location.href* как раз и будет соответствовать этому адресу.

В *location.href*можно записывать свои новые значения. Например, в данном примере кнопка загружает в текущее окно новую страницу:

```
<form>
<input type=button value="Yahoo"
  onClick="location.href='http://www.yahoo.com'; ">
</form>
```

Методические рекомендации к выполнению задания 2

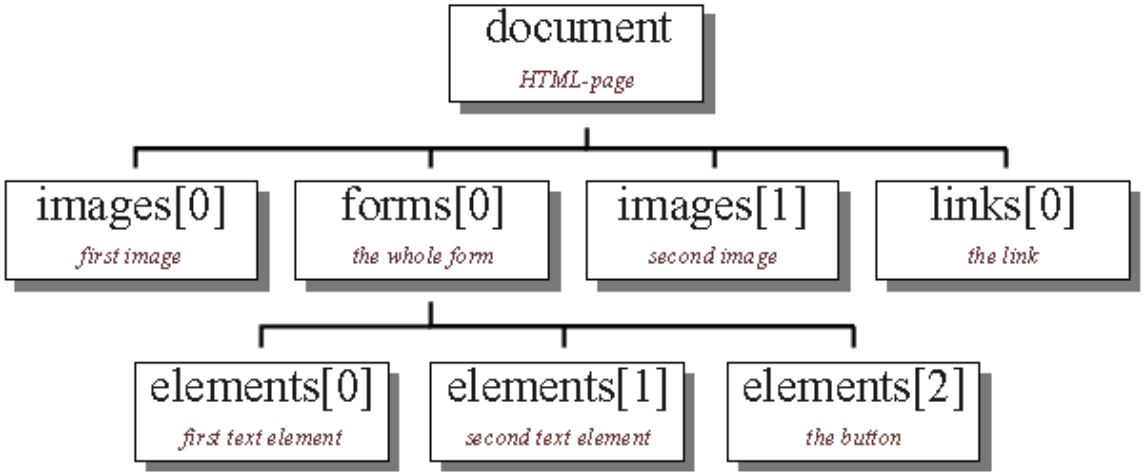
1. Создайте два произвольных рисунка формата gif. Сохраните их в отдельной папке на рабочем столе под названиями home.gif и ruler.gif.
2. Создайте HTML-страницу:

```
<html>
<head>
</head>
<body bgcolor=#925142>
<center>
<img src="home.gif" name="pic1" width=600 height=100>
</center>
<p>
<form name="myForm">
```

```
Name:
<input type="text" name="name" value=""><br>
e-Mail:
<input type="text" name="email" value=""><br><br>
<input type="button" value="Нажми" name="myButton"
onClick="alert('Спасибо!')">
</form>
<p>
<center>
<img src="ruler.gif" name="pic4" width=300 height=15>
<p>
<a href="http://rummelplatz.uni-mannheim.de/~skoch/">Моястраничка</a>
</center>
</body>
</html>
```

3. Проверьте результат.

Итак, мы имеем два рисунка, одну ссылку и некую форму с двумя полями для ввода текста и одной кнопкой. На следующем рисунке иллюстрируется иерархия объектов, создаваемая HTML-страницей из нашего примера:



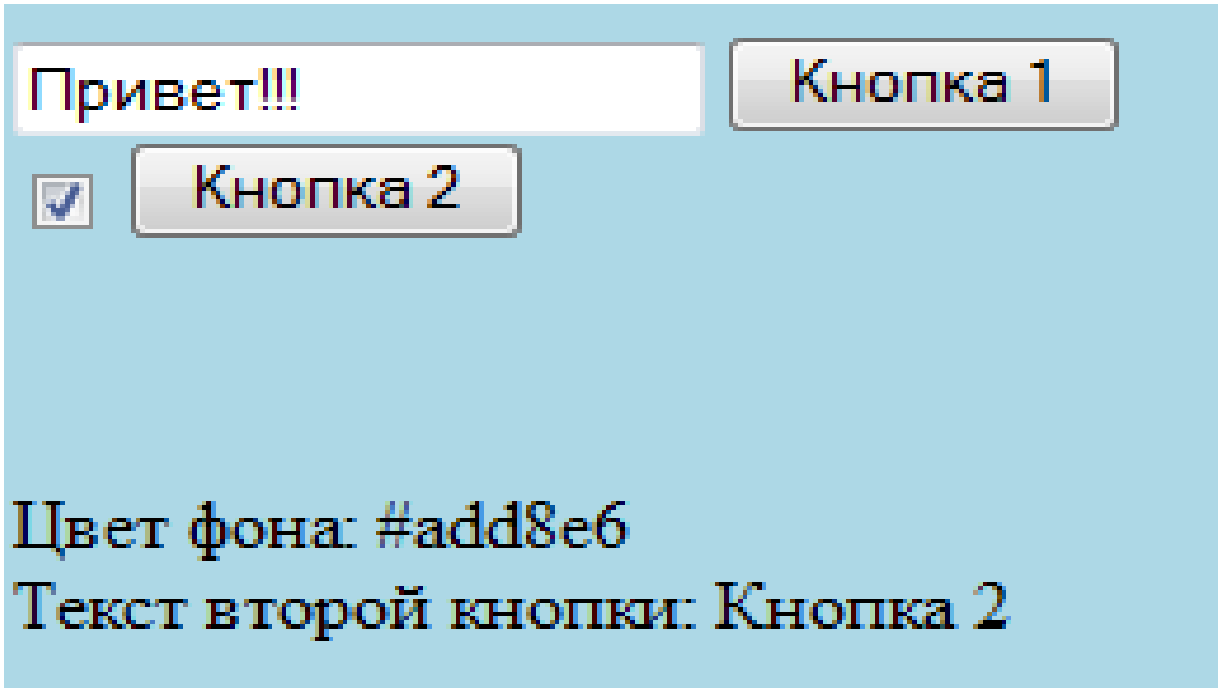
Методические рекомендации к выполнению задания 3

1. Создайте исходный код скрипта:

```
<html>
<head>
<title>Objects</title>
<script language="JavaScript">
<!-- hide
function first() {
// создает выпадающее окно, где размещается
// текст, введенный в поле формы
alert("Текст: " +  document.myForm.myText.value); }
function second() {
// данная функция проверяет состояние переключателей
varmyString= "Переключатель ";
// переключатель включен, или нет?
if (document.myForm.myCheckbox.checked) myString+= "включен"
    elsemyString+= "отключен";
// вывод строки на экран
alert(myString); }
// -->
</script>
</head>
<body bgcolor=lightblue>
<form name="myForm">
<input type="text" name="myText" value="Привет!!!">
<input type="button" name="button1" value="Кнопка 1"
  onClick="first()">
<br>
<input type="checkbox" name="myCheckbox" CHECKED>
<input type="button" name="button2" value="Кнопка 2"
  onClick="second()">
</form>
```

```
<p><br><br>
<script language="JavaScript">
<!-- hide
document.write("Цветфона: ");
document.write(document.bgColor + "<br>");
document.write("Текст второй кнопки: ");
document.write(document.myForm.button2.value);
// -->
</script>
</body>
</html>
```

3. Сравните результат



Вопросы для самоконтроля:

1 Каким образом представляются объекты на веб-странице?
2 Что понимается под иерархией объектов?
3 Для чего предназначен объект document?
4 Что относится к свойствам объекта document?
5 Приведите примеры объектов HTML.
6 Каким образом организуется управление иерархией объектов?
7 Какой скрипт нужно написать для того, чтобы узнать текст, введенный читателем?
8 Какие функции выполняет объект location?
9 Для чего предназначен объект window?
10 К какому виду объектов принадлежит окно браузера с точки зрения JavaScript?

Лабораторное занятие № 3

Тема: Создание фреймов

Цель занятия: Повторить приемы создания документов, содержащих фреймы. Ознакомиться с принципами использования адресации для фреймов

Задания:

1 Ознакомиться с теоретическими аспектами темы.

2 Создайте веб-страницу с несколькими ссылками в одном фрейме.

3 Создайте веб-страницу, в которой в одном фрейме создайте несколько ссылок, но если посетитель активирует какую-либо из них, соответствующая страница будет помещена не в тот же самый фрейм, а в соседний.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер.

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

В общем случае окно браузера может быть разбито в несколько отдельных фреймов. Это означает, что фрейм определяется как некое выделенное в окне браузера поле в форме прямоугольника. Каждый из фреймов выдает на экран содержимое собственного документа. Таким образом, можно, к примеру, создать два фрейма. В первый такой фрейм загрузить "домашнюю страницу" фирмы Netscape, а во второй - фирмы Microsoft. Для создания фреймов необходимы два тэга: <frameset> и <frame>. HTML-страница, создающая два фрейма, в общем случае может выглядеть следующим образом:

```
<html>
<frameset rows="50%,50%">
<frame src="page1.htm" name="frame1">
<frame src="page2.htm" name="frame2">
</frameset>
</html>
```

В результате будут созданы два фрейма. Во фрейме <frameset> мы используем свойство *rows*. Это означает, два фрейма будут расположены друг над другом. В верхний фрейм будет загружена HTML-страница *page1.htm*, а в нижнем фрейме разместится документ *page2.htm*.

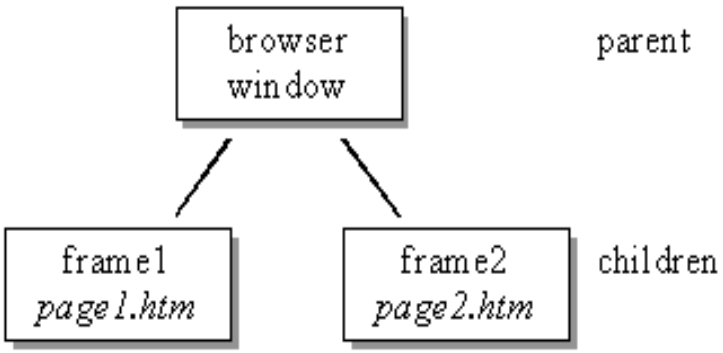
Если нужно, чтобы документы располагались не друг над другом, а рядом, то следует в тэге <frameset> писать не *rows*, а *cols*. Фрагмент *"50%,50%"* сообщает, насколько велики должны быть оба получившихся окна. Вы имеете также возможность записать *"50%,*"*, если не хотите утруждать себя расчетами, насколько велик должен быть второй фрейм, чтобы в сумме получалась все те же 100%. Вы можете также задать размер фрейма в пикселях, для чего достаточно после числа не ставить символ %.

Любому фрейму можно присвоить уникальное имя, воспользовавшись в тэге <frame> атрибутом *name*. Такая возможность пригодится в языке JavaScript для доступа к фреймам.

При создании web-страниц можно использовать несколько вложенных

тэгов <frameset>.

JavaScript организует все элементы, представленные на web-странице, в виде некой иерархической структуры. То же самое относится и к фреймам. На следующем рисунке показана иерархия объектов, представленных в первом примере:



В вершине иерархии находится окно браузера (browser window). В данном случае он разбито на два фрейма. Таким образом, окно, как объект, является родоначальником, родителем данной иерархии (parent), а два фрейма - соответственно, его потомки (children). Мы присвоили этим двум фреймам уникальные имена - *frame1* и *frame2*. И с помощью этих имен можно обмениваться информацией с двумя указанными фреймами.

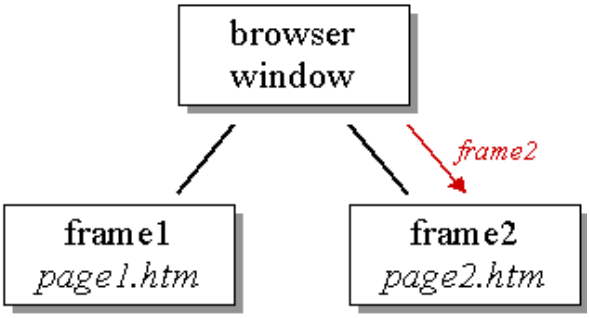
С помощью скрипта можно решить следующую задачу: допустим посетитель активирует некую ссылку в первом фрейме, однако соответствующая страница должна загружаться не в этот же фрейм, а в другой. Примером такой задачи может служить составление меню (или навигационных панелей), где один фрейм всегда остается неизменным, но предлагает посетителю несколько различных ссылок для дальнейшего изучения данного сайта.

Чтобы решить эту задачу, нужно рассмотреть на три случая:

- главное окно/фрейм получает доступ к фрейму-потомку
- фрейм-потомок получает доступ к родительскому окну/фрейму
- фрейм-потомок получает доступ к другому фрейму-потомку

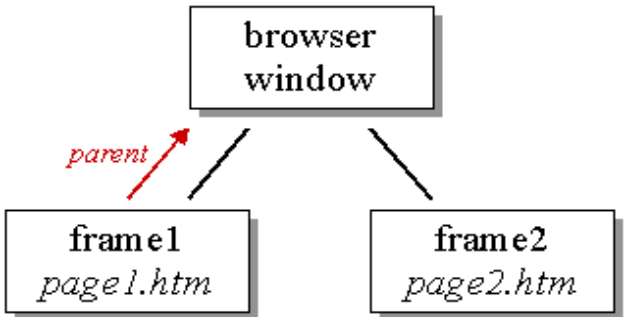
С точки зрения объекта "окно" (window) два указанных фрейма называются *frame1* и *frame2*. Как можно видеть на предыдущем рисунке, существует прямая взаимосвязь между родительским окном и каждым фреймом. Так образом, если Вы пишете скрипт для родительского окна - то есть для страницы, создающей эти фреймы - то можете обращаться к этим фреймам, просто называя их по имени. Например, можно написать:

frame2.document.write("Это сообщение передано от родительского окна.");



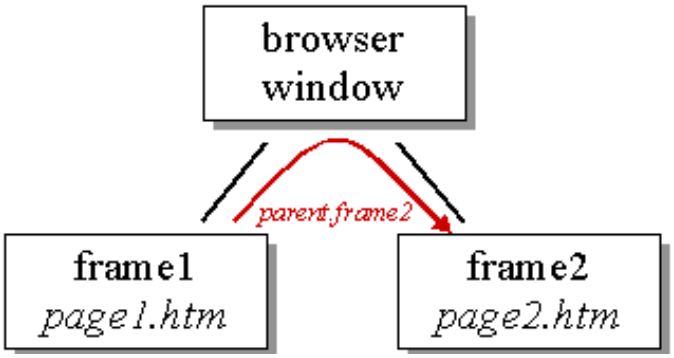
В некоторых случаях понадобится, находясь во фрейме, получать доступу к родительскому окну. Например, это бывает необходимо, если нужно при следующем переходе избавиться от фреймов. В таком случае удаление фреймов означает лишь загрузку новой страницы вместо содержавшей фреймы. В нашем случае это загрузка страницы в родительское окно. Сделать это нам поможет доступ к родительскому- *parent* - окну (или родительскому фрейму) из фреймов, являющихся его потомками. Чтобы загрузить новый документ, мы должны внести в *location.href* новый адрес URL. Поскольку нужно избавиться от фреймов, следует использовать объект location из родительского окна. (Напомним, что в каждый фрейм можно загрузить собственную страницу, то мы имеем для каждого фрейма собственный объект location). Итак, можно загрузить новую страницу в родительское окно с помощью команды:

parent.location.href= "http://...";



И наконец, очень часто Вам придется решать задачу обеспечения доступа с одного фрейма-потомка к другому такому же фрейму-потомку. Итак, как можно, находясь в первом фрейме, записать что-либо во второй - то есть, которой командой следует воспользоваться на HTML-странице *page1.htm*? Как можно увидеть на рисунке, между двумя этими фреймами нет никакой прямой связи. И потому мы не можем просто так вызвать *frame2*, находясь в фрейме *frame1*. С точки же зрения родительского окна второй фрейм действительно существует и называется *frame2*, а к самому родительскому окну можно обратиться из первого фрейма по имени *parent*. Таким образом, чтобы получить доступ к объекту document, разместившемуся во втором фрейме, мы должны написать следующее,:

parent.frame2.document.write("Привет, это вызов из первого фрейма.");



Предположим, мы имеем три фрейма с именами *frame1*, *frame2* и *frame3*.Допустим посетитель активирует ссылку в *frame1*. Нужно, чтобы при этом в два других фрейма загружались две различные web-страницы. В качестве решения этой задачи Вы можете, например, воспользоваться функцией:

```
function loadtwo() {
  parent.frame1.location.href= "first.htm";
  parent.frame2.location.href= "second.htm";}
```

Если же Вы хотите сделать функцию более гибкой, то можете воспользоваться возможностью передачи переменной в качестве аргумента. Результат будет выглядеть как:

```
function loadtwo(url1, url2) {
parent.frame1.location.href= url1;
parent.frame2.location.href= url2;}
```

Послеэтогоможноорганизоватьвызовфункции: loadtwo("first.htm", "second.htm") или loadtwo("third.htm", "forth.htm"). Очевидно, передача аргументов делает Вашу функцию более гибкой. В результате Вы можете использовать ее многократно и в различных контекстах.

Методические рекомендации к выполнению задания 2

1 Создайте документ html.

2 Создайте произвольную html-страничку и сохраните ее в отдельной папке на рабочем столе под названием cell.htm.

3 Создайте второй html-документ. Введите:

```
<frameset cols="50%,50%">
<frameset rows="50%,50%">
<frame src="cell.html">
<frame src="cell.html">
</frameset>
<frameset rows="33%,33%,33%">
<frame src="cell.html">
<frame src="cell.html">
<frame src="cell.html">
</frameset>
</frameset>
```

Можно задать толщину границы между фреймами, воспользовавшись в

тэге <frameset> параметром *border*. Запись border=0 означает, что между тэгами нет какой-либо границы.

Методические рекомендации к выполнению задания 3

1 В одном фрейме создайте несколько ссылок. Однако, если посетитель активирует какую-либо из них, соответствующая страница будет помещена не в тот же самый фрейм, а в соседний. Сперва необходимо написать скрипт, создающий указанные фреймы:

```
<html>
<frameset rows="80%,20%">
<frame src="start.htm" name="main">
<frame src="menu.htm" name="menu">
</frameset>
</html>
```

Здесь *start.htm* - это та страница, которая первоначально будет показана в главном фрейме (*main*). У нас нет никаких специальных требований к содержимому этой страницы.

2 Сохраните документ под названием Frames.htm.

3 Создайте три документа с произвольным содержанием: first.htm, second.htm, third.htm.

4 Создайте документ ***menu.htm***:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function load(url) {
  parent.main.location.href= url;
}
// -->
</script>
</head>
<body>
<a href="javascript:load('first.htm')">first</a>
<a href="second.htm" target="main">second</a>
<a href="third.htm" target="_top">third</a>
</body>
</html>
```

В данном примере использованы несколько способов загрузки новой страницы во фрейм *main*. В первой ссылке для этой цели используется функция *load()*. Давайте посмотрим, как это делается:

```
<a href="javascript:load('first.htm')">first</a>
```

Вместо явной загрузки новой страницы сдесь предлагается браузеру выполнить некую команду на языке JavaScript - для этого используется параметр *javascript:* вместо обычного *href*. Далее, внутри скобок можно увидеть *'first.htm'*. Эта строка передается в качестве аргумента функции *load()*. Сама же

функция *load()* определяется следующим образом:

```
function load(url) {
parent.main.location.href= url;}
```

Здесь внутри скобок написано *url*. Это означает, что строка 'first1.htm' при вызове функции заносится в переменную *url*. И эту новую переменную теперь можно использовать при работе внутри функций *load()*.

Во второй ссылке присутствует параметр *target*. Это одна из конструкций языка HTML. Как видно, здесь всего лишь указывается имя необходимого фрейма. Заметим, что в этом случае не обязательно ставить перед именем указанного фрейма слово *parent*. Причина такого отступления от правил в том, что параметр *target* - это функция языка HTML, а не JavaScript.

На примере третьей ссылки можно увидеть, как с помощью *target* можно избавиться от фреймов. Если нужно избавиться от фреймов с помощью функции *load()*, то необходимо написать в ней лишь parent.location.href= url.

Вопросы для самоконтроля:

1 Что называют фреймом?
2 Какую роль в веб-документе играют фреймы?
3 Какие теги необходимы для создания фреймов?
4 Какие атрибуты и параметры можно задать относительно фреймов?
5 Можно ли присвоить имя фреймам?
6 Каким образом организуется обеспечение доступа с одного фрейма-потомка к другому такому же фрейму-потомку?
7 Для чего предназначена функция loadtwo?

Лабораторное занятие № 4

Тема: Окна и динамическое управление документами

Цель: Ознакомиться с принципами работы с окнами и динамическим управлением документами, научить использовать свойства окон при создании веб-страниц и создавать документы, изменяющие свойства других документов

Задания:

1 Ознакомиться с теоретическими аспектами темы.
2 Создайте веб-страницу, в которой в новое окно с помощью метода open() записывается другая страница.
3 Создайте веб-страницу, в которой производится создание нового окна фиксированного размера.
4 Создать документ с использованием методов объекта window.
5 Создать документ с использованием команд генерации нового документа.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

Открытие новых окон в браузере - грандиозная возможность языка JavaScript. Можно либо загружать в новое окно новые документы, либо (динамически) создавать новые материалы. Также можно управлять самим процессом создания окна. Например, можно указать, должно ли новое окно иметь строку статуса, панель инструментов или меню. Кроме того, можно задать размер окна. Список свойств окна, которыми можно управлять:

Directories	Yes / no
Height	количество пикселов
Location	Yes / no
Menubar	Yes / no
Resizable	Yes / no
Scrollbars	Yes / no
Status	Yes / no
Toolbar	Yes / no
Width	количество пикселов

Как видите, открывая окна, мы должны использовать три аргумента:

myWin= open("cell.htm", "displayWindow", "width=400, height=300, status=no, toolbar=no, menubar=no");

Второй аргумент - это имя окна. Если оно известно, то можно загрузить туда новую страницу с помощью записи

<a href="cell.html" target="displayWindow">

При этом необходимо указать имя соответствующего окна (если же такого окна не существует, то с этим именем будет создано новое).

myWin - это не имя окна, но только с помощью этой переменной можно получить доступ к окну. И поскольку это обычная переменная, то область ее действия - лишь тот скрипт, в котором она определена. Имя окна (в данном случае это *displayWindow*) - уникальный идентификатор, которым можно пользоваться с любого из окон браузера.

Чтобы закрыть окно понадобится метод close(). Более того, можно таким же образом создавать и другие документы Web, такие как VRML-сцены и т.д. Для удобства можно размещать эти документы в отдельном окне или фрейме.

Методические рекомендации к выполнению задания 2

1 Создайте новый документ cell.html произвольного содержания.

2 Создайте новый документ html, скрипт показан ниже.

```
<html>
<head>
<script language="JavaScript">
<!-- hide
functionopenWin() {
  myWin= open("cell.htm");
```

```
}
// -->
</script>
</head>
<body>
<form>
<input type="button" value="Открытьновоеокно" onClick= "openWin()" >
</form>
</body>
</html>
```

В представленном примере в новое окно с помощью метода open() записывается страница *cell.htm*.

Методические рекомендации к выполнению задания 3

1. Создайте новый документ html

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function openWin2() {
  myWin= open("cell.htm", "displayWindow",
 "width=400,height=300,status=no,toolbar=no,menubar=no,scrollbar=no");}
// -->
</script>
</head>
<body>
<form>
<input type="button" value="Открытьновоеокно" onClick="openWin2()">
</form>
</body>
</html>
```

В этом скрипте открывается новое окно размером 400x300 пикселей. Оно не имеет ни строки статуса, ни панели инструментов, ни меню. Как видите, свойства окна формулируются в строке

"width=400,height=300,status=no,toolbar=no,menubar=no".

Обратите внимание также и на то, что не следует помещать в этой строке символы пробела.

Методические рекомендации к выполнению задания 4

1. Создайтедокумент:

```
<html>
<script language="JavaScript">
<!-- hide
function closeIt() {
  close();}
```

```
// -->
</script>
<center>
<form>
<input type=button value="Close it" onClick="closeIt()">
</form>
</center>
</html>
```

Если теперь в новом окне нажать на кнопку, то окно будет закрыто. open() и close() - это методы объекта window. Следует писать не просто *open()* и *close()*, а *window.open()* и *window.close()*. В нашем случае объект *window* можно опустить - нет необходимости писать префикс window, если нужно вызвать один из методов этого объекта (и такое возможно только для этого объекта).

Методические рекомендации к выполнению задания 5

1. Создайте HTML-документ:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function openWin3() {
  myWin= open("", "displayWindow",
    "width=500,height=400,status=yes,toolbar=yes,menubar=yes");
// открыть объект document для последующей печати
  myWin.document.open();
    // генерировать новый документ
myWin.document.write("<html><head><title>On-the-fly");
  myWin.document.write("</title></head><body>");
  myWin.document.write("<center><font size=+3>");
  myWin.document.write("This HTML-document has been created ");
  myWin.document.write("with the help of JavaScript!");
  myWin.document.write("</font></center>");
  myWin.document.write("</body></html>");
// закрыть документ - (но не окно!)
myWin.document.close(); }
// -->
</script>
</head><body><form>
<input type=button value="On-the-fly" onClick="openWin3()">
</form>
</body>
</html>
```

Давайте рассмотрим функцию openWin3(). Очевидно, мы сначала открываем новое окно браузера. Поскольку первый аргумент функции open() - пустая строка (""), то это значит, что мы не желаем в данном случае указывать

конкретный адрес URL. Браузер должен создать дополнительно новый документ. В скрипте мы определяем переменную *myWin*. И с ее помощью можем получать доступ к новому окну. Обратите внимание, что в данном случае нельзя воспользоваться для этой цели именем окна (*displayWindow*). После того, как открыли окно, наступает очередь открыть для записи объект document. Делается это с помощью команды:

```
// открыть объект document для последующей печати
myWin.document.open();
```

Здесь мы обращаемся к open() - методу объекта document. Но это совсем не то же самое, что метод open() объекта window. Эта команда не открывает нового окна - она лишь готовит document к предстоящей печати. Кроме того, мы должны поставить перед *document.open()* приставку *myWin*, чтобы получить возможность писать в новом окне. В последующих строках скрипта с помощью вызова *document.write()* формируется текст нового документа:

```
// генерироватьновыйдокумент
myWin.document.write("<html><head><title>On-the-fly");
myWin.document.write("</title></head><body>");
myWin.document.write("<center><font size=+3>");
myWin.document.write("This HTML-document has been created ");
myWin.document.write("with the help of JavaScript!");
myWin.document.write("</font></center>");
myWin.document.write("</body></html>");
```

Как видно, здесь мы записываем в документ обычные тэги языка HTML. То есть мы фактически генерируем разметку HTML. При этом можно использовать абсолютно любые тэги HTML. По завершении этого нужно вновь закрыть документ. Это делается следующей командой:

```
// закрыть документ - (но не окно!)
myWin.document.close();
```

Вопросы для самоконтроля:

1 Перечислите свойства окна, которыми можно управлять.

2 Какие три аргумента нужно использовать при открытии окна?

3 При помощи какой переменной осуществляется доступ к окну?

4 Какой метод применяется для закрытия окна?

5 При помощи какой команды осуществляется открытие объекта document для последующей печати?

6 При помощи какой команды осуществляется закрытие документа?

7 При помощи какой команды осуществляется закрытие окна?

Лабораторное занятие № 5

Тема: Управление окнами

Цель занятия: Изучить приемы управления окнами в JavaScript

Задания:

1 Ознакомиться с теоретическим материалом.

2 Создайте документ, открывающий новое окно браузера и загружающий в него другую страничку с помощью метода open().

3 Создайте документ с использованием команд создания окна размерами 250x100 пикселов, которое не имеет ни строки статуса, ни панель инструментов, ни меню, ни полосы прокрутки.

4 Создайте документ, который открывает новое окно.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

Открытие окон

Вы можете создавать окно с помощью метода Window. Следующее утверждение создает окно *"msgWindow"*, которое показывает содержание файл *sesame.html*:

MsgWindow=window.open("sesame.html")

Следующее утверждение создает окно *"homeWindow"*, которое показывает домашнюю страницу Netscape:

homeWindow=window.open("http://www.netscape.com")

Окна могут иметь два названия. Когда вы создаете окно, имя окна не требуется. Но если нужно обратиться к окну из другого окна, то окно должно иметь свое уникальное имя (в данном случае *"displayWindow"*).

Заметим, что вы имеете возможность управлять самим процессом создания окна. Например, вы можете указать, должно ли новое окно иметь строку статуса, панель инструментов, меню или полосы прокрутки. Кроме того, вы можете задать размер окна.

Вы можете закрывать окна с помощью языка *JavaScript*. Следующие утверждения закрывают текущее окно:

Window.close ()

Self.close ()

Замечание: не использовать следующее утверждение в обработке результата close()

Следующее утверждение закрывает окно, названное *"msgWindow"*:

MsgWindow.close()

Методические рекомендации к выполнению задания 2

Создайте страницу *sesame.html* с произвольным содержанием. Сохраните документ. Создайте новый документ в той же папке, впишите в него

следующий скрипт. Этот скрипт открывает новое окно браузера и загружает в него страничку *sesame.html* с помощью метода open():

```
<html>
<head>
<scriptlanguage="JavaScript">
<!-- hide
functionopenWin() {
msgWindow= open("sesame.html")   }
 // -->
</script>
</head>
<body>
<form>
<input type="button" value="Открытьновоеокно"   onClick="openWin()">
</form>
</body>
</html>
```

Методические рекомендации к выполнению задания 3

Создайте документ, введите в него скрипт, представленный ниже. В данном примере рассматривается создание окна размерами 250x100 пикселов, которое не имеет ни строки статуса, ни панель инструментов, ни меню, ни полосы прокрутки.

```
<html>
<head>
<script language="JavaScript">
<!-- hide
   function openWin() {
    msgWindow= open("sesame.html",displayWindow",
     "width=250,height=100,status=no,
    toolbar=no,menubar=no,scrollbars=no")    }
  // -->
</script>
</head>
<body>
<form>
<input type="button" value="Открытьновоеокно"
  onClick="openWin()">
</form>
</body>
</html>
```

Как видно в примере, свойства окна записаны в строке *"width=250,height=100,status=no,toolbar=no,menubar=no,scrollbars=no"*, в которой следует обратить внимание на то, что вам не следует помещать в ней символы пробела.

Методические рекомендации к выполнению задания 4

Создайте документ, который открывает новое окно. Загрузите туда web-страницу, где при нажатии кнопки оно будет закрыто. Скриптдокумента:

```
<html>
<script language="JavaScript">
<!-- hide
  function closeIt(){
close();   }
 // -->
</script>
<center>
<form>
<input type=button value="Закрытьокно" onClick="closeIt()">
</form>
</center>
</html>
```

open() и close() - это методы объекта window. Мы должны помнить, что следует писать не просто open() и close(), а window.open() и window.close(). Однако в нашем случае объект window можно опустить - Вам нет необходимости писать префикс window, если Вы хотите всего лишь вызвать один из методов этого объекта (и такое возможно только для этого объекта).

Вопросы для самоконтроля:

1 Могут ли окна иметь более одного названия?

2 При помощи каких команд осуществляется открытие нового окна браузера и загрузка в него страницы?

3 Каким образом осуществляется задание параметров открываемого окна?

Лабораторное занятие № 6

Тема: Строка состояния и таймеры

Цель: Изучить приемы создания строк состояния и таймеров

Задания:

1 Ознакомиться с теоретическими сведениями.

2 Создайте документ, содержащий две кнопки, которые можно использовать, чтобы записывать некий текст в строку состояния и, соответственно, затем его стирать.

3 Создайте документ с использованием процедур onMouseOver и onMouseOut.

4 Создайте документ, демонстрирующий кнопку, которая открывает выпадающее окно не сразу, а по истечении 3 секунд.

5 Создайте документ с бегущей строкой в строке состояния.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

Строка состояния

Составленные программы на JavaScript могут выполнять запись в строку состояния - прямоугольник в нижней части окна браузера. Все, что необходимо для этого сделать - всего лишь записать нужную строку в *window.status*.

Механизм вывода текста в строку состояния удобно использовать при работе со ссылками. Вместо того, чтобы выводить на экран URL данной ссылки, можно просто на словах объяснять, о чем будет говориться на следующей странице.

Таймеры

С помощью функции Timeout (или таймера) можно запрограммировать компьютер на выполнение некоторых команд по истечении некоторого времени.

Прокрутка

Можно запрограммировать прокрутку в основной линейке. Рассмотрим также и всевозможные усовершенствования этой линейки. Создать бегущую строку довольно просто. Сперва мы должны записать в строку состояния некий текст. Затем по истечении короткого интервала времени мы должны записать туда тот же самый текст, но при этом немного переместив его влево. Если мы это сделаем несколько раз, то у пользователя создастся впечатление, что он имеет дело с бегущей строкой.

Однако при этом нужно помнить еще и о том, что обязаны каждый раз вычислять, какую часть текста следует показывать в строке состояния (как правило, объем текстового материала превышает размер строки состояния).

Скроллинг используется в Интернет довольно широко. Возможен вариант, когда одна часть текста приходит слева, а другая - справа. И когда они встречаются посередине, то в течение некоторых секунд текст остается неизменным.

Методические рекомендации к выполнению задания 2

1 Создайте документ:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function statbar(txt) {
  window.status = txt;}
// -->
</script>
</head>
<body>
```

```
<form>
<input type="button" name="look" value="Писать!"
onClick="statbar('Привет! Это окно состояния!');">
<input type="button" name="erase" value="Стереть!"
   onClick="statbar('');">
</form>
</body>
</html>
```

В этом примере создаются две кнопки, которые можно использовать, чтобы записывать некий текст в строку состояния и, соответственно, затем его стирать.

Обе эти кнопки вызывают функцию statbar(). Вызов от клавиши *Писать!* выглядит следующим образом: statbar('Привет! Это окно состояния!');

В скобках написана строка: *'Привет! Это окно состояния!'*. Это как раз и будет текст, передаваемый функции statbar(). В свою очередь, можно видеть, что функция statbar() определена следующим образом:

```
function statbar(txt) {
  window.status = txt;}
```

В заголовке функции в скобках мы поместили слово *txt*. Это означает, что строка, которую мы передали этой функции, помещается в переменную *txt*. Передача функциям переменных - прием, часто применяемый для придания функциям большей гибкости. Вы можете передать функции несколько таких аргументов - необходимо лишь отделить их друг от друга запятыми. Строка *txt* заносится в строку состояния посредством команды window.status = txt.

Соответственно, удаление текста из строки состояния выполняется как запись в *window.status* пустой строки.

Методические рекомендации к выполнению задания 3

1 Создайте документ dontclck.htm с произвольным содержанием.

2 Создайте второй документ:

```
<ahref="dontclck.htm"
 onMouseOver="window.status='Don't click me!'; return true;"
 onMouseOut="window.status='';">ссылка</a>
```

3 Сохраните документ под названием Main.htm.

4 Просмотрите результат, запустив в окне браузера документ Main.htm.

В этом примере достаточно лишь поместить указатель вашей мыши над ссылкой и в строке состояния появится соответствующий текст.

Здесь используются процедуры *onMouseOver* и *onMouseOut*, чтобы отслеживать моменты, когда указатель мыши проходит над данной ссылкой. В onMouseOver нужно возвращать результат *true* для того, чтобы браузер не выполнял свой собственный код обработки события MouseOver. Как правило, в строке состояния браузер показывает URL соответствующей ссылки. Если же мы не возвратим значение *true*, то сразу же после того, как наш код был выполнен, браузер перепишет строку состояния на свой лад - то есть наш текст

будет тут же затерт и читатель не сможет его увидеть.

В некоторых случаях нужно печатать символы кавычек. Например, нужно напечатать текст *Don't click me* - однако поскольку нужно передать эту строку в процедуру обработки события onMouseOver, то для этого используются одинарные кавычки. Между тем, как слово *Don't* тоже содержит символ одинарной кавычки. И в результате если написать *'Don't ...'*, браузер запутается в этих символах. Чтобы разрешить эту проблему, достаточно лишь поставить обратный слэш \ перед символом кавычки - это означает, что данный символ предназначен именно для печати. (То же самое можно делать и с двойными кавычками).

Методические рекомендации к выполнению задания 4

1 Создайтедокумент:

```
<script language="JavaScript">
<!-- hide
function timer() {
  setTimeout("alert('Времяистекло!')", 3000);}
// -->
</script>
...
<form>
<input type="button" value="Timer" onClick="timer()">
</form>
```

В следующем скрипте демонстрируется кнопка, которая открывает выпадающее окно не сразу, а по истечении 3 секунд. Здесь *setTimeout()* - это метод объекта window. Он устанавливает интервал времени. Первый аргумент при вызове - это код JavaScript, который следует выполнить по истечении указанного времени. В нашем случае это вызов - *"alert('Время истекло!')"*. Обратите пожалуйста внимание, что код на JavaScript должен быть заключен в кавычки.

Во втором аргументе компьютеру сообщается, когда этот код следует выполнять. При этом время нужно указывать в миллисекундах (3000 миллисекунд = 3 секундам).

Методические рекомендации к выполнению задания 5

1 Введите скрипт:

```
<html>
<head>
<scriptlanguage="JavaScript">
<!-- hide
// define the text of the scroller
var scrtxt = "Это JavaScript! " +
    "Это JavaScript! " +
    "Это JavaScript!";
var len = scrtxt.length;
```

```
var width = 100;
var pos = -(width + 2);
function scroll() {
  // напечатать заданный текст справа и установить таймер
  // перейти на исходную позицию для следующего шага
  pos++;
  // вычленить видимую часть текста
  var scroller = "";
if (pos == len) {
   pos = -(width + 2);
}
  // если текст еще не дошел до левой границы, то мы должны
  // добавить перед ним несколько пробелов. В противном случае мы
должны
  // вырезать начало текста (ту часть, что уже ушла за левую границу
if (pos < 0) {
   for (var i = 1; i <= Math.abs(pos); i++) {
    scroller = scroller + " ";}
   scroller = scroller + scrtxt.substring(0, width - i + 1);  }
  else {
   scroller = scroller + scrtxt.substring(pos, width + pos);  }
// разместить текст в строке состо\яни\я
  window.status = scroller;
  // вызвать эту функцию вновь через 100 миллисекунд
setTimeout("scroll()", 100);}
// -->
</script>
</head>
<bodyonLoad="scroll()">
Это пример прокрутки в строке состояния средствами JavaScript.
</body>
</html>
```

Большая часть функции scroll() нужна для вычленения той части текста, которая будет показана пользователю. Чтобы запустить процесс прокрутки, используется процедура обработки события onLoad, описанная в тэге <body>. То есть функция *scroll()* будет вызвана сразу же после загрузки HTML-страницы.

Через посредство процедуры onLoad вызывается функция scroll(). Первым делом в функции scroll() устанавливается таймер. Этим гарантируется, что функция scroll() будет повторно вызвана через 100 миллисекунд. При этом текст будет перемещен еще на один шаг и запущен другой таймер. Так будет продолжаться без конца.

Вопросы для самоконтроля:

1 Для чего предназначена строка состояния?

2 Можно ли осуществлять управление строкой состояния?

3 Каким образом осуществляется вывод текста в строке состояния?

4 При помощи какой функции можно запрограммировать компьютер на выполнение некоторых команд по истечении некоторого времени?

5 Можно ли запрограммировать прокрутку в основной линейке?

Лабораторное занятие № 7

Тема: Предопределенные объекты

Цель: Научить работать с предопределенными объектами: Date, Array, Math

Задания:

1 Ознакомиться с теоретическим материалом.

2 Создайте документ, печатающий текущую дату и время.

3 Создайте документ, создающий на экране изображение работающих часов.

4 Создайте документ, использующий массив.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

В JavaScript разрешено пользоваться некоторыми заранее заданными объектами. Примерами таких объектов могут служить Date, Array или Math. Объект Date позволяет работать как со временем, так и с датой. Например, можно легко определить, сколько дней еще остается до следующего рождества. Или можете внести в HTML-документ запись текущего времени.

Рассмотрим пример, который высвечивает на экран текущее время. Сперва мы должны создать новый объект Date. Для этого используем оператором *new*:

today= new Date()

Здесь создается новый объект Date, с именем *today*. Если при создании этого нового объекта Date не указаны какие-либо определенные дата и время, то будут предоставлены текущие дата и время. То есть, после выполнения команды today= new Date() вновь созданный объект *today* будет указывать именно те дату и время, когда данная команда была выполнена.

Объект Date предоставляет нам кое-какие методы, которые теперь могут применяться к нашему объекту *today*. Например, этометоды - getHours(), setHours(), getMinutes(), setMinutes(), getMonth(), setMonth() итакдалее. Обратите пожалуйста внимание, что объект Date лишь содержит определенную запись о дате и времени. Он не уподобляется часам, автоматически отслеживающим время каждую секунду, либо миллисекунду. Чтобы зафиксировать какое-либо другие дату и время, мы можем

воспользоваться видоизмененным конструктором (это будет метод Date(), который при создании нового объекта Date вызывается через оператор new):

today= new Date(1997, 0, 1, 17, 35, 23)

При этом будет создан объект Date, в котором будет зафиксировано первое января 1997 года 17:35 и 23 секунд. Таким образом, выбираются дата и время по следующему шаблону:

Date(year, month, day, hours, minutes, seconds)

Для обозначения января необходимо использовать число 0, а не 1. Число 1 будет обозначать февраль, ну и так далее.

Массивы играют в программировании очень важную роль. Массив может быть полезен там, где имеется много взаимосвязанных переменных. При этом к каждой из них можно получить доступ, воспользовавшись общим названием и неким номером. Допустим, есть массив в именем *names*. В этом случае мы можем получить доступ к первой переменной с именем name, написав *names[0]*. Вторая переменная носит *name[1]* и так далее. Начиная с версии 1.1 языка JavaScript (Netscape Навигатор 3.0), можно использовать объект Array. Можно создать новый массив, записав myArray= new Array(). После этого можно начать заносить в массив значения:

myArray[0]= 17;

myArray[1]= "Stefan";

myArray[2]= "Koch";

Массивы JavaScript обладают большой гибкостью. Например, нет нужды беспокоиться о размере массива - он устанавливается динамически. Если написать myArray[99]= "xyz", размер массива будет установлен 100 элементов.

Не имеет значения, заносятся ли в массив числа, строки, либо другие объекты. Если необходимо в скрипте выполнять математические расчёты, то некоторые полезные методы для этого можно найти в объекте Math. Например, имеется метод random(). Напишем функцию, позволяющую генерировать случайные числа. Теперь, чтобы работать на всех без исключения платформах, нам не нужно ничего, кроме метода random().

Если вызвать функцию Math.random(), то получите случайное число, лежащее в диапазоне между 0 и 1. Один из возможных результатов вызова document.write(Math.random()) (при каждой новое загрузке данной страницы здесь будет появляться другое число):

0.9070647660301312

Методические рекомендации к выполнению задания 2

1 Создайте документ, печатающий текущую дату и время.

```
<script language="JavaScript">
<!-- hide
now= new Date();
document.write("Time: "+ now.getHours()+":" + now.getMinutes() + "<br>");
document.write("Date: " + (now.getMonth() + 1) + "/" + now.getDate() + "/" +
```

```
(now.getYear()));
// -->
</script>
```

Результат будет выглядеть примерно следующим образом:

Time: 20:40

Date: 10/24/2009

Здесь используются такие методы, как getHours(), чтобы вывести на экран время и дату, указанные в объекте Date с именем *now*. Помните также, что нужно увеличивать на единицу значение, получаемое от метода getMonth().

В данном скрипте не выполняется проверки на тот случай, если количество минут окажется меньше, чем 10. Это значит, что выводится запись времени примерно в следующем виде: *14:3*, что на самом деле должно было бы означать *14:03*.

Методические рекомендации к выполнению задания 3

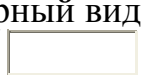
1 Создайте документ, создающий на экране изображение работающих часов:

```
<html>
<head>
<script Language="JavaScript">
<!-- hide
var timeStr, dateStr;
function clock() {
      now= new Date();
// время
      hours= now.getHours();
      minutes= now.getMinutes();
      seconds= now.getSeconds();
      timeStr= "" + hours;
      timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;
      timeStr+= ((seconds < 10) ? ":0" : ":") + seconds;
      document.clock.time.value = timeStr;
      // дата
      date= now.getDate();
      month= now.getMonth()+1;
      year= now.getYear();
      dateStr= "" + month;
      dateStr+= ((date < 10) ? "/0" : "/") + date;
      dateStr+= "/" + year;
      document.clock.date.value = dateStr;
      Timer= setTimeout("clock()",1000);}
// -->
</script>
</head>
<body onLoad="clock()">
```

```
<form name="clock">
Время:
<input type="text" name="time" size="8"  value=""><br>
Дата:
<input type="text" name="date" size="8" value="">
</form>
</body>
</html>
```

2 Просмотрите результат.

Примерный вид результата:

Время: 

Дата: 

Здесь для ежесекундной коррекции времени и даты используется метод setTimeout(). Фактически это сводится к кому, что каждую секунду создается новый объект Date, в который заносится текущее время.

Можно видеть, что функции clock() вызываются программой обработки события onLoad, помещенной в тэг <body>. В разделе body HTML-страницы имеется два элемента формы для ввода текста. Функция clock() записывает в оба эти элемента в корректном формате текущие время и дату. Для этой цели используются две строки *timeStr* и *dateStr*. Существует проблема с индикацией, когда количество минут меньше 10 - в данном скрипте эта проблема решается с помощью следующей строки: timeStr+= ((minutes < 10) ? ":0" : ":") + minutes;

Как видим, количество минут заносится в строку *timeStr*. Если у нас менее 10 минут, то мы еще должны приписать спереди 0. Эта строка в скрипте может показаться немного странной, и ее можно было бы переписать в более знакомом Вам виде:

```
if (minutes < 10) timeStr+= ":0" + minutes
  else timeStr+= ":" + minutes;
```

Методические рекомендации к выполнению задания 4

1. Создайте документ, использующий массив:

```
<scriptlanguage="JavaScript">
<!-- hide
myArray= new Array();
myArray[0]= "first element";
myArray[1]= "second element";
myArray[2]= "third element";
for (var i= 0; i< 3; i++) {
  document.write(myArray[i] + "<br>");}
// -->
</script>
```

Данный скрипт печатает следующий текст:

first element

second element

third element

Первым делом создается здесь новый массив с именем myArray. Затем заносятся в него три различных значения. После этого мы запускаем цикл, который трижды выполняет команду document.write(myArray[i] + "
");. В переменной *i* ведется отсчет циклов от 0 до 2. Заметим, что в цикле мы пользуемся конструкцией *myArray[i]* . И поскольку *i* меняет значения от 0 до 2, то в итоге мы получаем три различных вызова document.write(). Иными словами, мы могли бы расписать этот цикл как:

```
document.write(myArray[0] + "<br>");
document.write(myArray[1] + "<br>");
document.write(myArray[2] + "<br>");
```

Вопросы для самоконтроля:

1 Примерами каких объектов служат объекты Date, Array, Math?
2 Для чего предназначен объект Date?
3 Что называют массивом?
4 Для чего предназначены массивы?
5 Каким образом осуществляется доступ к массиву?
6 Каким образом производится установка размера массива?
7 Для чего предназначен объект Math?
8 Перечислите методы объекта Date.
9 Каким образом производится создание объектов Date, Array, Math?
10 Какой объект позволяет работать со временем?

Лабораторное занятие № 8

Тема: Создание форм

Цель: Изучить приемы и методы создания форм и работы с ними

Задания:

1 Ознакомиться с теоретическими аспектами темы.
2 Создайте документ, содержащий два элемента для ввода текста.
3 Создайте документ, содержащий текстовое поле и кнопку, при нажатии на которую осуществляется перевод курсора в начало текстового поля.
4 Создайте документ, содержащий текстовое поле и кнопку, при нажатии на которую осуществляется перевод курсора в начало текстового поля и выделение содержимого данного текстового поля.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

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.

Проверка на присутствие определенных символов

В некоторых случаях нужно 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 или 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.

Предоставление информации, введенной в форму

Самый простой способ состоит в передаче данных формы по электронной почте. Если нужно, чтобы за вносимыми в форму данными следил сервер, то необходимо использовать интерфейс CGI (Common Gateway Interface). Последнее позволяет 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.

Выделение определенного элемента формы

С помощью метода focus() можно сделать форму более дружественной. Так, можно выбрать, который элемент будет выделен в первую очередь. Либо можно приказать браузеру выделить ту форму, куда были введены неверные данные. То есть, что браузер сам установит курсор на указанный Вами элемент формы, так что пользователю не придется щелкать по форме, прежде чем что-либо занести туда.

Методические рекомендации к выполнению задания 2

1 Создайте документ, содержащий два элемента для ввода текста.

```
<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">
ВведитеВашеимя:<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"
onClick="test2(this.form)">
</body>
</html>
```

В первый элемент пользователь должен вписать свое имя, во второй элемент - адрес для электронной почты. Можно ввести туда какую-нибудь информацию и нажать клавишу. Попробуйте также нажать клавишу, не введя в форму никакой информации. Что касается информации, введенной в первый элемент, то будет выводиться 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 сложный. Попробуйте ввести простую строку – например, свое 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.

В разделе body создаются 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. Однако к команде if мы добавлен символ ||. Он 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 строка пуста, либо в ней отсутствует символ @. (Второй оператор в команде if следит за тем, чтобы введенная строка содержала @).

Методические рекомендации к выполнению задания 3

1 Создайте документ, содержащий текстовое поле и кнопку, при нажатии на которую осуществляется перевод курсора в начало текстового поля. Для осуществления перехода курсора используйте функцию:

```
functionsetfocus() {
document.first.text1.focus();}
```

Эта запись может 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*.

Методические рекомендации к выполнению задания 4

1 Создайте документ, содержащий текстовое поле и кнопку, при нажатии на которую осуществляется перевод курсора в начало текстового поля и выделение содержимого данного текстового поля. При выполнении задания используйте функцию:

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

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

Вопросы для самоконтроля:

1 Каким образом осуществляется проверка информации, введенной в

формулу?

2 Каким образом производится проверка на отсутствие определенных символов?

3 Для чего предназначен метод focus()?

4 Что нужно сделать для того, чтобы за вносимыми в форму данными следил сервер?

Лабораторное занятие № 9

Тема: Объекты Image

Цель: Научить создавать объекты типаImage

Задания:

1 Ознакомиться с теоретическими материалами темы.

2 Создайте два рисунка Image1.jpeg и Image2.jpeg. Создайте документ, в котором содержится первый рисунок. При нажатии на кнопку первый рисунок должен быть заменен вторым, имеющим те же размеры, что и первый.

3 Создайте документ, в котором загружается определенная картинка, при наведении на которую происходит смена картинку на другую.

4 Создайте документ, содержащий три рисунка (image1.jpeg, image2.jpeg, image3.jpeg), служащих ссылками на документы (link1.htm, link2.htm, link3.htm), при наведении на определенный рисунок, рисунок должен изменяться, при нажатии, должна запускаться соответствующая страничка.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

Изображения на web-странице

Объект Image стал доступен начиная с версии с 1.1 языка JavaScript (то есть с Netscape Navigator 3.0). С помощью объекта Image можно 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 полный эффект.

В JavaScript 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, можно определить, какой 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 пикселах).

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

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

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

Загрузка новых изображений

Часто приходится смену изображений на web-странице и для этого требуется использование атрибута *src*. Как и в случае тэга <img>, атрибут *src* содержит адрес представленного изображения. Теперь - в языке JavaScript 1.1 - имеется 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 уже нельзя 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. Решением данной проблемы была бы упреждающая загрузка изображения. Для этого мы должны создать новый объект Image. Рассмотрим следующие строки:

hiddenImg= newImage();

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.

Браузер должен к моменту запроса закончить упреждающую загрузку, чтобы необходимое изображение было показано без задержки. Поэтому, если Нужно 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 показанными командами. Мы лишь пытаемся чуть раньше загрузить изображение - поэтому и пользователь может увидеть их раньше. В результате и весь процесс пройдет более гладко. Можно создавать красивые эффекты, используя смену изображений в качестве реакции на определенные события. Например, можно изменять изображения в тот момент, когда курсор мыши попадает на определенную часть страницы.

Объект Image дает возможность создавать действительно сложные эффекты. Однако заметим, что *не всякое* изображение или программа JavaScript способны улучшить страницу. Не количество изображений делает web-страницу привлекательной, а их качество. Сама загрузка 50 килобайт плохой графики способна вызвать раздражение. При создании специальных эффектов с изображениями с помощью JavaScript следует помнить об этом.

Методические рекомендации к выполнению задания 2

1 Создайте два рисунка Image1.jpeg и Image2.jpeg. Создайте документ, в котором содержится первый рисунок. При нажатии на кнопку первый рисунок должен быть заменен вторым, имеющим те же размеры, что и первый. Для создания такого документа используйте следующие команды:

```
<html>
<head>
</head>
<body bgcolor=#925142>
<center>
<img src="image1.jpeg" name="myImage" width=400 height=400>
</center>
<p>
<form name="myForm">
<input type="button" value="Нажми" name="myButton" onClick= 'document.myImage.src= "image2.jpeg";' >
</form>
</center>
```

```
</body>
</html>
```

Методические рекомендации к выполнению задания 3

1 Создайте документ, в котором загружается определенная картинка, при наведении на которую происходит смена картинку на другую.

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>
```

Методические рекомендации к выполнению задания 4

1 Заранее подготовьте рисунки: image1.jpeg, image2.jpeg, image3.jpeg.

2 Создайте три документа с произвольным содержанием: link1.htm, link2.htm, link3.htm.

3 Создайте документ, содержащий три рисунка (image1.jpeg, image2.jpeg, image3.jpeg), служащих ссылками на документы (link1.htm, link2.htm, link3.htm), при наведении на определенный рисунок, рисунок должен изменяться, при нажатии, должна запускаться соответствующая страничка.

Вопросы для самоконтроля:

1 Для чего предназначен объект Image?

2 Какие действия можно производить с использованием объекта Image?

3 В виде чего представляются все изображения в JavaScript?

4 Какие команды используются при загрузке новых изображений?

5 Каким образом производится замена рисунков при нажатии?

Лабораторное занятие № 10

Тема: Использование слоев

Цель: Изучить основные понятия технологии слоев

Задание:

1 Ознакомиться с теоретическими сведениями по теме.

2 Создайте новый документ, содержащий два слоя. В первом из них поместите изображение, а во втором - текст. Все, что нужно сделать - показать этот текст поверх данного изображения.

3 Создайте документ с использованием команд показа и скрытия слоев.

4 Создайте документ с использованием функции перемещения слоев.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

Использование **слоев** позволяет выполнять точное позиционирование таких объектов web-страницы, как изображения. Кроме того, можно перемещать объекты по вашей HTML-странице. Вы можете также делать объекты невидимыми. Управлять слоями можно легко с помощью языка JavaScript.

Создание слоев

Чтобы создать слой, мы должны использовать либо тэг <layer> либо <ilayer>. Вы можете воспользоваться следующими параметрами:

name="*layerName*"	Название слоя
left=*xPosition*	Абсцисса левого верхнего угла
top=*yPosition*	Ордината левого верхнего угла
z-index=*layerIndex*	Номер индекса для слоя
width=*layerWidth*	Ширина слоя в пикселах
clip="*x1_offset, y1_offset, x2_offset, y2_offset"*	Задает видимую область слоя
above="*layerName*"	Определяет, какой слой окажется под нашим
below="*layerName*"	Определяется, какой слой окажется над нашим
Visibility=show|hide|inherit	Видимость этого слоя
bgcolor="*rgbColor*"	Цвет фона - либо название стандартного цвета, либо rgb-запись
background="*imageURL*"	Фоновая картинка

Тэг <layer> используется для тех слоев, которые Вы можете точно позиционировать. Если же Вы не указываете положение слоя (с помощью параметров *left* и *top*), то по умолчанию он помещается в верхний левый угол окна.

Тэг <ilayer> создает слой, положение которого определяется при формировании документа.

Рассмотрим теперь, как можно получить доступ к слоям через JavaScript. Начнем же мы с примера, где пользователь получает возможность, нажимая кнопку, прятать или показать некий слой.

Для начала мы должны знать, каким образом слои представлены в JavaScript. Как обычно, для этого имеются несколько способов. Самое лучшее - дать каждому слою свое имя. Так, если мы задаем слой

<layer ... name=myLayer>

...

</layer>

то в дальнейшем можем получить доступ к нему с помощью конструкции *document.layers["myLayer"]*. Согласно документации, предоставляемой фирмой Netscape, мы можем также использовать запись *document.myLayer* - однако в моем браузере это приводит к сбою. Конечно, это всего лишь проблема предварительной версии и в заключительном варианте будет успешно решена (сейчас я пользуюсь Netscape Navigator 4.0 PR3 на WinNT). Однако, по-видимому, нет никаких проблем с конструкцией *document.layers["myLayer"]* - поэтому мы и будем пользоваться именно такой альтернативой из всех возможных.

Доступ к этим слоям можно также получить через целочисленный индекс. Так, чтобы получить доступ к самому нижнему слою, Вы можете написать *document.layers[0]*. Обратите внимание, что индекс - это не то же самое, что параметр z-index. Если, например, Вы имеете два слоя, называемые *layer1* и *layer2* с номерами z-index 17 и 100, то Вы можете получить доступ к этим слоям через *document.layers[0]* и *document.layers[1]*, а **не** через *document.layers[17]* и *document.layers[100]*.

Перемещение слоев

Свойства *left* и *top* определяют положение данного слоя. Вы можете менять его, записывая в эти атрибуты новые значения. Например, в следующей строке задается горизонтальное положение слоя в 200 пикселов:

document.layers["myLayer2"].left= 200;

Методические рекомендации к выполнению задания 2

Создайте новый документ, содержащий два слоя. В первом из них поместите изображение, а во втором - текст. Все, что нужно сделать - показать этот текст поверх данного изображения.

Текст поверх изображения

Исходный код:

```
<html>
<layer name=pic z-index=0 left=200 top=100>
<img src="img.gif" width=160 height=120>
</layer>
<layer name=txt z-index=1 left=200 top=100>
<font size=+4><i> Layers-Demo </i></font>
</layer>
</html>
```

Как видим, с помощью тэга <layer> мы формируем два слоя. Оба слоя позиционируются как 200/100 (через параметры *left* и *top*). Все, что находится

между тэгами <layer> и </layer> (или тэгами <ilayer> и </ilayer>) принадлежит описываемому слою.

Кроме того, мы используем параметр *z-index*, определяя тем самым порядок появления указанных слоев - то есть, в нашем случае, Вы тем самым сообщаете браузеру, что текст будет написан поверх изображения. В общем случае, именно слой с самым высоким номером z-index будет показан поверх всех остальных. Вы не ограничены в выборе z-index лишь значениями 0 и 1 - можно выбирать вообще любое положительное число.

Так, если в первом тэге <layer> Вы напишете *z-index=100*, то текст окажется под изображением - его слой номер Z-индекса (z-index=1). Вы сможете увидеть текст сквозь изображение, поскольку я использовал в нем прозрачный фон (формат gif89a).

Текст под изображением

Методические рекомендации к выполнению задания 3

Создайте документ, содержащий скрипт:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function showHide() {
  if (document.layers["myLayer"].visibility == "show")
    document.layers["myLayer"].visibility= "hide"
  else document.layers["myLayer"].visibility= "show";}
// -->
</script>
</head>
<body>
<ilayer name=myLayer visibility=show>
<font size=+1 color="#0000ff"><i>This text is inside a layer</i></font>
</ilayer>
<form>
<input type="button" value="Show/Hide layer" onClick="showHide()">
</form></body></html>
```

Даннаякнопкавызываетфункцию*showHide()*. Можно видеть, что в этих функциях реализуется доступ к такому свойству объекта layer (*myLayer*), как **видимость**. Присвоивая параметру *document.layers["myLayer"].visibility* значения *"show"* или *"hide"*, Вы можете показать или скрыть наш слой.

Заметим, что *"show"* и *"hide"* - это строки, а не зарезервированные ключевые слова, то есть Вы **не можете** написать *document.layers["myLayer"].visibility= show*.

Вместо тэга <layer> я также пользовался тэгом <ilayer>, поскольку хотел поместить этот слой в "информационный поток" документа.

Методические рекомендации к выполнению задания 4

Создайте документ с использованием функции перемещения слоев. Скрипт выглядит следующим образом:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function move() {
  if (pos < 0) direction= true;
  if (pos > 200) direction= false;
  if (direction) pos++
    else pos--;
  document.layers["myLayer2"].left= pos;
}
// -->
</script>
</head>
<body onLoad="setInterval('move()', 20)">
<ilayer name=myLayer2 left=0>
<font size=+1 color="#0000ff"><i>This text is inside a layer</i></font>
</ilayer>
</body>
</html>
```

Мы создаем слой с именем *myLayer2*. Можно видеть, что в тэге <body>мы пользуемся процедурой *onLoad*. Нам необходимо начать прокручивание слоя, как только страница будет загружена. В процедуре обработки события *onLoad* мы пользуемся функцией *setInterval()*. Это один из новых методов версии 1.2 языка JavaScript (то есть версии JavaScript, реализованной в Netscape Navigator 4.0). Им можно пользоваться, чтобы вызывать некую функцию мвновь и вновь через определенные интервалы времени. В прошлом для этого мы пользовались функцией *setTimeout()*. Функция *setInterval()* работает почти так же, однако Вам нужно вызвать ее всего лишь один раз.

С помощью *setInterval()* мы вызываем функцию *move()* каждые 20 миллисекунд. А функция *move()*, в свою очередь, всякий раз смещает слой на новую позицию. И поскольку мы вызываем эту функцию вновь и вновь, то мы получаем быстрый скроллинг нашего текста. Все, что мы нужно сделать в функции *move()* - это вычислить новую координату для слоя и записать ее: *document.layers["myLayer2"].left= pos*.

Если Вы посмотрите исходный код этой части в онлайновом описании, то увидите, что в действительности мой код выглядит несколько иначе - я добавил некий фрагмент кода с тем, чтобы люди, работающие со старыми версиями JavaScript-браузеров, не получали из-за этого никаких сообщений об ошибках. Как этого можно достичь? Следующий фрагмент кода будет выполняться только на тех браузерах, которые воспринимают язык JavaScript 1.2:

```
<script language="JavaScript1.2">
<!-- hide
document.write("Вы используете браузер, совместимый с JavaScript 1.2.");
// -->
</script>
```

Та же самая проблема возникает, когда мы работаем с объектом Image. Мы можем аналогичным способом переписать кодекс. Установка переменной *browserOK* решает эту проблему.

Следующий пример демонстрирует, как может осуществляться перекрывание слоев:

This text is inside a layer

Вопросы для самоконтроля:

1 Какие возможности предоставляет использование слоев?

2 Каким образом производится создание слоев?

3 Для чего используется тег <layer>?

4 Какими командами нужно воспользоваться для того, чтобы получить доступ к слою через целочисленный индекс?

5 При помощи каких команд производится определение положения данного слоя?

6 В каких случаях используется параметр z-index?

7 В чем отличия тэгов <layer> и <ilayer>?

8 Каким образом производится перемещение слоев?

9 Каким образом производится перекрывание слоев?

10 Для чего предназначено событие onLoad?

Лабораторное занятие № 11

Тема: Работа со слоями

Цель: Изучить приемы вырезки из слоя, создания вложенных слоев, использования различных эффектов с прозрачными слоями

Задания:

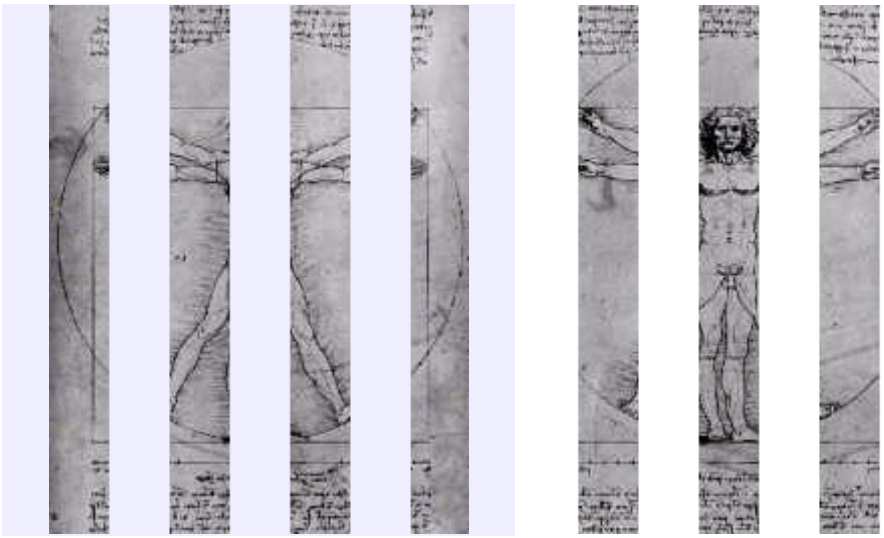
1 Ознакомиться с теоретическими аспектами темы.

2 Создать документ согласно методическим указаниям.

3 Создайте документ, содержащий три кнопки, которые могут запускать и останавливать движение слоев.

4 Создайте документ, в котором можно задать выделяемую область. В документе должен быть использован механизм вырезания и перемещение изображения. Это нужно для того, чтобы вырезаемая часть была зафиксирована, т.е. чтобы при перемещении всего изображения не происходила смена видимого на экране фрагмента.

5 Создайте документ, в котором используются два изображения (сплошные серые зоны здесь на самом деле являются прозрачными):



Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

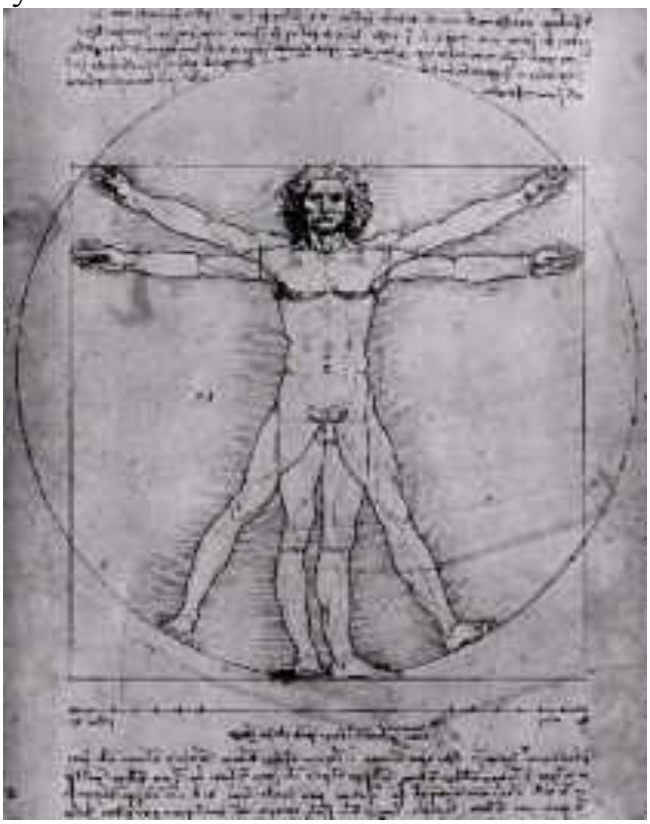
Методические рекомендации к выполнению задания 1

Можно постулировать, что какая-то (прямоугольная) часть слоя будет нам видима. Все же, что лежит за ее пределами, показано на экране не будет. Такой прием называется вырезанием. Например, в разметке HTML можно задать следующую функцию **вырезания**:

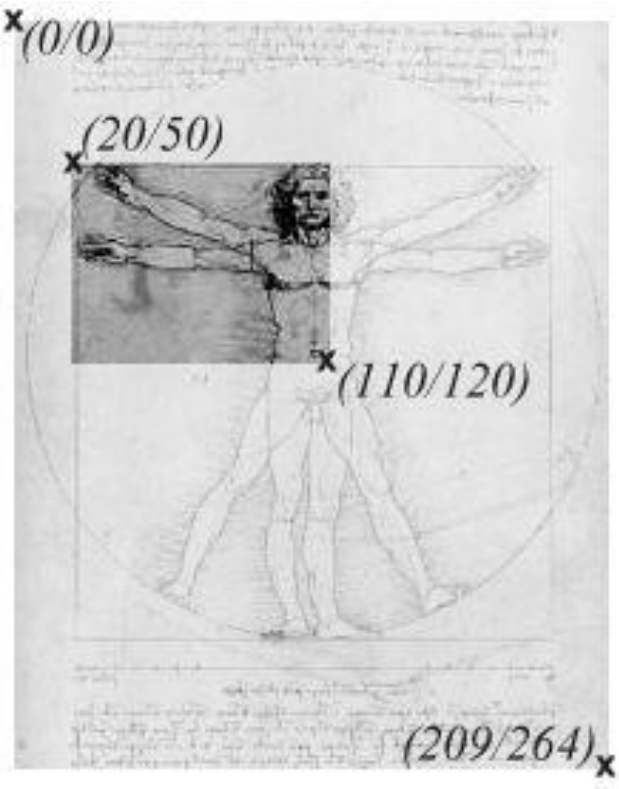
```
<ilayer left=0 top=0 clip="20,50,110,120">
<img src="davinci.jpg" width=209 height=264>
</ilayer>
```

(Здесь приписаны параметры *left=0* и *top=0*, поскольку в противном случае, если этого не сделать, то в некоторых версиях возникают проблемы).

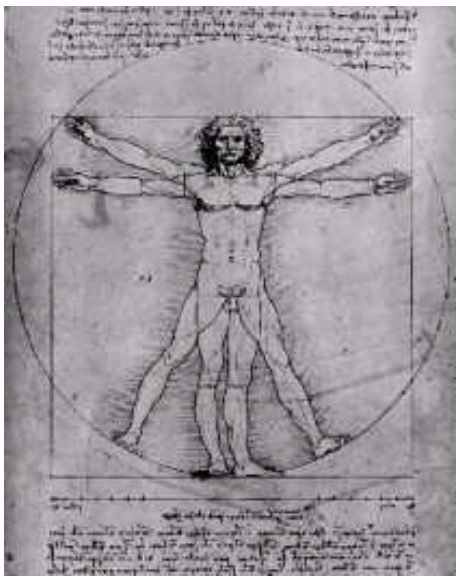
Хотя само изображение и имеет размеры 209x264 пикселов, мы можем видеть лишь его малую часть:



Данный фрагмент изображения имеет размер 90x70 (пикселов). Первые два значения, указанные в атрибуте *clip* (атрибуте HTML-тэга <layer> или <ilayer>), указывают верхний левый угол вырезаемой части. Следующие два значения указывают нижний правый угол. Сказанное можно проиллюстрировать следующим рисунком:



Еще более интересных результатов можно добиться, управляя вырезанной частью с помощью языка JavaScript. Точнее, Вы можете изменять значения свойств *clip.left*, *clip.top*, *clip.right* и *clip.bottom* объекта Layer. Достаточно всего лишь занести в одно из этих свойств новое значение, как фрагмент тут же будет кадрирован соответствующим образом . В следующем примере параметры вырезанной части изображения меняются динамически, и в результате у пользователя создается впечатление, будто изображение медленно "растет":



Методические рекомендации к выполнению задания 2

Создайте документ со следующим кодом скрипта:

<html>

<head>

```
<script language="JavaScript">
<!-- hide
var middleX, middleY, pos;
functionstart() {
  // получитьразмеризображени\я
var width= document.layers["imgLayer"].document.davinci.width;
  var height= document.layers["imgLayer"].document.davinci.height;
// определить, какой пиксел находитс\я в центре изображени\я
middleX= Math.round(width/2);
  middleY= Math.round(height/2);
// начальная позици\я
  pos= 0;
  // запуск!
  show();}
function show() {
  // увеличить размер вырезаемой области
pos+= 2; // step size
  document.layers["imgLayer"].clip.left= middleX- pos;
  document.layers["imgLayer"].clip.top= middleY- pos;
  document.layers["imgLayer"].clip.right= middleX+ pos;
  document.layers["imgLayer"].clip.bottom= middleY+ pos;
// проверить, не высвечено ли все изображение
if (!((pos > middleX) && (pos > middleY)))
    setTimeout("show()", 20);  }
// -->
</script>
</head>
<body>
<ilayer name="imgLayer" clip="0,0,0,0">
<img name=davinci src="davinci.jpg" width=209 height=264>
</ilayer>
<form>
<input type=button value="Start" onClick="start()">
</form>
</body>
</html>
```

Кнопка, представленная в разделе *<body>*, вызывает функцию *start()*. Сначала мы должны определить точку, с которой нам следует начать работу - фактически это будет некий пиксел в центре нашего изображения. Значения координат x и y этого пиксела мы помещаем в переменные *middleX* и *middleY*. После этого мы вызываем функцию *show()*, которая задает размеры вырезаемой части изображения в зависимости от значений переменных *middleX, middleY* и параметра *pos*. При этом значение переменной *pos* автоматически увеличивается при каждом вызове функции *show()*. То есть размер вырезаемой части изображения с каждым разом становится все больше и больше. В самом

конце процедуры *show()* мы устанавливаем таймер с помощью вызова *setTimeout()* - и благодаря этому функция *show()* вызывается вновь и вновь. И этот процесс остановится только тогда, когда изображение будет показано целиком.

Заметим, что размер изображения мы получаем в самом начале функции *start()*:

```
var width= document.layers["imgLayer"].document.davinci.width;
 var height= document.layers["imgLayer"].document.davinci.height;
```

С помощью конструкции *document.layers["imgLayer"]* мы можем обратиться к слою с именем *imgLayer*. Однако почему после *document.layers["imgLayer"]* мы ставим *document*? Дело в том, что каждый слой имеет свою собственную HTML-страницу - то есть, **каждый слой имеет свой объект document**. Чтобы получить доступ к изображению внутри слоя *imgLayer*, нам необходимо получить доступ к этому объекту document. В приведенном выше примере такое изображение носило название *davinci*. Все остальное поле листа должно быть чистым.

Методические рекомендации к выполнению задания 3

Как мы уже видели, слой может содержать несколько различных объектов. Он могут даже включать в себя другие слои. Конечно, может возникнуть вопрос, для чего это нужно. На самом деле есть несколько причин, чтобы пользоваться вложенными слоями. Рассмотрим несколько примеров, демонстрирующих применение вложенных слоев.

В первом примере используется слой (называемый *parentLayer*), в который вложено еще два других слоя (*layer1* и *layer2*).

Создайте документ, содержащий три кнопки, которые могут запускать и останавливать движение слоев. Исходный код скрипта:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
// начальна\япозици\я
var pos0= 0;
var pos1= -10;
var pos2= -10;
// движение?
var move0= true;
var move1= false;
var move2= false;
// направление?
var dir0= false;
var dir1= false;
var dir2= true;
function startStop(which) {
```

```
  if (which == 0) move0= !move0;
  if (which == 1) move1= !move1;
  if (which == 2) move2= !move2;}
 function move() {
  if (move0) {
   // перемещение parentLayer
   if (dir0) pos0--
    else pos0++;
   if (pos0 < -100) dir0= false;
   if (pos0 > 100) dir0= true;
   document.layers["parentLayer"].left= 100 + pos0;      }
  if (move1) {
   // перемещение parentLayer
   if (dir1) pos1--
    else pos1++;
   if (pos1 < -20) dir1= false;
   if (pos1 > 20) dir1= true;
   document.layers["parentLayer"].layers["layer1"].top= 10 + pos1;  }
  if (move2) {
   // перемещение parentLayer
   if (dir2) pos2--
    else pos2++;
   if (pos2 < -20) dir2= false;
   if (pos2 > 20) dir2= true;
   document.layers["parentLayer"].layers["layer2"].top= 10 + pos2;      }}
 // -->
 </script>
 </head>
 <body onLoad="setInterval('move()', 20)">
 <ilayer name=parentLayer left=100 top=0>
 <layer name=layer1 z-index=10 left=0 top=-10>
 Этопервыйслой
 </layer>
 <layer name=layer2 z-index=20 left=200 top=-10>
 Это второй слой
 </layer>
 <br><br>
  Это главный (родительский) слой
 </ilayer>
 <form>
 <input type="button" value="Move/Stop parentLayer"
onClick="startStop(0);">
 <input type="button" value="Move/Stop layer1" onClick="startStop(1);">
 <input type="button" value="Move/Stop layer2" onClick="startStop(2);">
 </form>
```

```
</body>
</html>
```

Также можно видеть, что перемещение слоя *parentLayer* сопровождается перемещением и двух других слоев, тогда как перемещение слоя *layer1* (или *layer2*) ни на что другое не влияет. Этот пример демонстрирует возможность объединения группы объектов с помощью механизма вложенных слоев.

Можно видеть, что внутри *parentLayer* мы определили два слоя. Это как раз и есть вложенные слои. Как получить к этим слоям доступ в языке JavaScript? Как это делается, можно посмотреть в функции *move()*:

```
document.layers["parentLayer"].left= 100 + pos0;
  ...
 document.layers["parentLayer"].layers["layer1"].top= 10 + pos1;
  ...
 document.layers["parentLayer"].layers["layer2"].top= 10 + pos2;
```

Чтобыполучитьдоступквложеннымслоям, Вамнедостаточнобудетпростонаписать*document.layers["layer1"]*или*document.layers["layer2"]*, поскольку слои *layer1* и *layer2* лежат **внутри** *parentLayer*.

Методические рекомендации к выполнению задания 4

Создайте документ, в котором можно задать выделяемую область. В документе должен быть использован механизм вырезания и перемещение изображения. Это нужно для того, чтобы вырезаемая часть была зафиксирована, т.е. чтобы при перемещении всего изображения не происходила смена видимого на экране фрагмента.

Исходный код скрипта:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
var pos= 0; // начальноеположение
var direction= false;
function moveNclip() {
  if (pos<-180) direction= true;
 if (pos>40) direction= false;
  if (direction) pos+= 2
    else pos-= 2;
  document.layers["clippingLayer"].layers["imgLayer"].top= 100 + pos;}
// -->
</script>
</head>
<body onLoad="setInterval('moveNclip()', 20);">
<ilayer name="clippingLayer" z-index=0 clip="20,100,200,160" top=0 left=0>
<ilayer name="imgLayer" top=0 left=0>
<img name=davinci src="davinci.jpg" width=209 height=264>
```

```
</ilayer>
</ilayer>
</body>
</html>
```

И снова, можно видеть обращение к вложенному слою:

```
document.layers["clippingLayer"].layers["imgLayer"].top= 100 + pos;
```

Методические рекомендации к выполнению задания 5

Интересные эффекты могут быть созданы с помощью (частично) прозрачных слоев. Сочетание специально подобранных изображений с прозрачными областями может создавать совершенно потрясающий результат. Не все форматы изображений поддерживают работу с прозрачными частями. В настоящее время лучший из отвечающих этому условию форматов - gif89a. Большинство новых графических программ поддерживает этот формат. Помимо этого, в Internet доступны некоторые свободно распространяемые инструменты для работы с графикой. Новый формат изображений PNG также поддерживает эффект прозрачных частей изображения.

Скриптдокумента:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
var pos= 0; // starting position
var myTimer= null;
function move() {
  pos+= 3; // step size
  document.layers["clippingLayer"].layers["imgLayer1"].top= -264 + pos;
  document.layers["clippingLayer"].layers["imgLayer2"].top= 264 - pos;
  if (pos< 264) myTimer= setTimeout('move()', 10)
    else myTimer= null;}
function start() {
  if (myTimer== null) {
    pos= 0;
    move();
  }}
// -->
</script>
</head>
<body onLoad="move()">
<form>
<input type=button value="Start" onClick="start();">
</form>
<ilayer name="clippingLayer" z-index=0 clip="209,264" top=0 left=0>
<layer name="imgLayer1" top=-264 left=0>
<img name=davinci1 src="davinci1.gif"   width=209 height=264>
```

```
</layer>
<layer name="imgLayer2" top=264 left=1>
<img name=davinci2 src="davinci2.gif"   width=209 height=264>
</layer>
</ilayer>
</body></html>
```

Вопросы для самоконтроля:

1 Охарактеризуйте процесс вырезания.
2 Перечислите основные команды для работы с изображениями.
3 Какие параметры слоев существуют?
4 Каким образом осуществляется изменение прозрачности слоев?
5 Какую роль выполняет функция move()?

Лабораторное занятие № 12

Тема: Модель событий в JavaScript

Цель: Рассмотреть приемы работы с моделями событий

Задание:

1 Создайте документ с использованием функции вывода сообщения.

2 Создайте документ, в котором на экран выводится некое изображение. При щелчке над изображением клавишей мыши должно появиться окно сообщений, где показаны координаты той точки, где в этот момент находилась мышь.

3 Создать документ согласно методическим указаниям.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

В JavaScript 1.2 поддерживается обработка следующих событий:

Abort	Focus	MouseOut	Submit
Blur	KeyDown	MouseOver	Unload
Click	KeyPress	MouseUp	
Change	KeyUp	Move	
DblClick	Load	Reset	
DragDrop	MouseDown	Resize	
Error	MouseMove	Select	

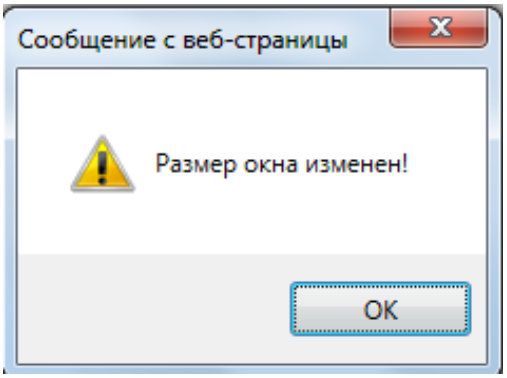
Изучая таблицу, можете увидеть, что была реализована обработка некоторых новых событий. На этом уроке мы и рассмотрим некоторые из них.

Сперва давайте рассмотрим событие *Resize*. С помощью этого события Вы можете определить, был бы размер окна изменен читателем.

Создайте новый документ со скриптом:

```
<html>
<head>
<script language="JavaScript">
window.onresize= message;
function message() {
alert("Размер окна изменен!"); }
</script>
</head>
<body>
Пожалуйста, измените размер этого окна.
</body>
</html>
```

При открытии окна в браузере появится строка «Пожалуйста, измените размер окна». Когда вы начнте изменять размер окна, то выйдет сообщение:



Встрокеwindow.onresize= message;

мы задаем процедуру обработки такого события. Точнее, функция *message()* будет вызываться всякий раз, как только пользователь изменит размер окна. JavaScript 1.2 ничего нового здесь не привносит. Например, если объект button, то можно определить процедуру обработки события следующим образом:

```
<form name="myForm">
<input type="button" name="myButton" onClick="alert('Click event occured!')">
</form>
```

Однако Вы можете написать это и по-другому:

```
<form name="myForm">
<input type="button" name="myButton">
</form>
...
<script language="JavaScript>
document.myForm.myButton.onclick= message;
```

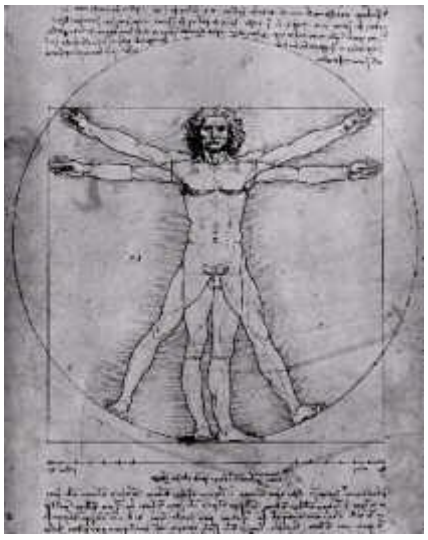
```
function message() {
  alert('Click event occured!'); }
</script>
```

Можно подумать, что вторая альтернатива немного сложнее. Однако почему тогда именно ее мы используем в первом скрипте? Причина состоит в том, что объект window нельзя определить через какой-либо определенный тэг - поэтому нам и приходится использовать второй вариант. Не следует писать *window.onResize* – имеется в виду, что нужно писать все прописными буквами; не нужно ставить после *message* скобки. Если написать *window.onresize= message()*, то браузер интерпретирует *message()* как вызов функции. Однако в нашем случае мы не хотим напрямую вызывать эту функцию - мы лишь хотим определить обработчик события.

Методические рекомендации к выполнению задания 2

В язык JavaScript 1.2 добавлен новый объект Event. Он содержит свойства, описывающие некое событие. Каждый раз, когда происходит какое-либо событие, объект Event передается соответствующей программе обработки.

Создайте документ, в котором на экран выводится некое изображение. При щелчке над изображением клавишей мыши должно появиться окно сообщений, где показаны координаты той точки, где в этот момент находилась мышь:



Кодскрипта:

```
<layer>
<a href="#" onClick="alert('x: ' + event.x + ' y: ' + event.y); return false;">
<img src="davinci.jpg" width=209 height=264 border=0></a>
</layer>
```

Как видите, в тэг *<a>* мы поместили программу обработки событий *onClick*, как это мы уже делали в предшествующих версиях JavaScript. Новое здесь заключается в том, что для создания окошка с сообщением мы используем *event.x* и *event.y*. А это как раз и есть объект Event, который здесь

нам нужен, чтобы узнать координаты мыши.

Все команды помещены в тэг <layer>. Благодаря этому мы получаем в сообщении координаты относительно данного слоя, т.е. в нашем случае относительно самого изображения. В противном же случае мы получили бы координаты относительно окна браузера (инструкция *return false;* используется здесь для того, чтобы браузер обрабатывал далее данную ссылку)

Объект Event получил следующие свойства:

Свойство	**Описание**
data	Массив адресов URL оставленных объектов, когда происходит событие *DragDrop*.
layerX	Горизонтальное положение курсора (в пикселах) относительно слоя. В комбинации с событием *Resize* это свойство представляет ширину окна браузера.
layerY	Вертикальное положение курсора (в пикселах) относительно слоя. В комбинации с событием *Resize* это свойство представляет высоту окна браузера.
modifiers	Строка, задающая ключи модификатора - ALT_MASK, CONTROL_MASK, META_MASK или SHIFT_MASK
pageX	Горизонтальное положение курсора (в пикселах) относительно окна браузера.
pageY	Вертикальное положение курсора (в пикселах) относительно окна браузера.
screenX	Горизонтальное положение курсора (в пикселах) относительно экрана.
screenY	Вертикальное положение курсора (в пикселах) относительно экрана.
target	Строка, представляющая объект, которому исходно было послано событие.
type	Строка, указывающая тип события.
which	ASCII-значение нажатой клавиши или номер клавиши мыши.
x	Синоним layerX
y	Синоним layerY

Методические рекомендации к выполнению задания 3

Одна из важных особенностей языка - перехват события. Если кто-то, к примеру, щелкает на кнопке, то вызывается программа обработки события *onClick*, соответствующая этой кнопке. С помощью обработки событий Вы можете добиться того, чтобы объект, соответсвующий вашему окну, документу или слою, перехватывал и обрабатывал событие еще до того, как для этой цели объектом указанной кнопки будет вызван обработчик событий. Точно так же объект вашего окна, документа или слоя может обрабатывать сигнал о событии еще до того, как он достигает своего обычного адресата.

Создайте новый документ со скриптом:

```
<html>
<head>
<script language="JavaScript">
window.captureEvents(Event.CLICK);
window.onclick= handle;
function handle(e) {
  alert("Объект window перехватывает это событие!");
  return true; // т.е. проследить ссылку}
</script>
</head>
<body>
<a href="test.htm">"Кликните" поэтойссылке.</a>
</body>
</html>
```

Как видно, мы не указываем программы обработки событий в тэге <a>. Вместо этого мы пишем

```
window.captureEvents(Event.CLICK);
```

с тем, чтобы перехватить событие *Click* объектом window. Обычно объект window не работает с событием *Click*. Однако, перехватив, мы затем его переадресуем в объект window.

Заметим, что в *Event.CLICK* фрагмент *CLICK* должен писаться заглавными буквами. Если же Вы хотите перехватывать несколько событий, то Вам следует отделить их друг от друга символами |. Например:

```
window.captureEvents(Event.CLICK | Event.MOVE);
```

Помимо этого в функции *handle()*, назначенной нами на роль обработчика событий, мы пользуемся инструкцией *return true;*. В действительности это означает, что браузер должен обработать и саму ссылку, после того, как завершится выполнение функции *handle()*. Если же Вы напишете вместо этого *return false;*, то на этом все и закончится.

Если теперь в тэге *<a>* Вы зададите программу обработки события *onClick*, то поймете, что данная программа при возникновении данного события вызвана уже не будет. И это не удивительно, поскольку объект window перехватывает сигнал о событии еще *до того*, как он достигает объекта link. Если же Вы определите функцию *handle()* как

```
function handle(e) {
  alert("Объект window перехватывает это событие!");
window.routeEvent(e);
  return true;}
```

то компьютер будет проверять, определены ли другие программы обработки событий для данного объекта. Переменная *e* - это наш объект Event, передаваемый функции обработки событий в виде аргумента.

Кроме того, Вы можете непосредственно послать сигнал о событии какому-либо объекту. Для этого Вы можете воспользоваться методом *handleEvent()*. Это выглядит следующим образом:

```
<html>
```

```
<script language="JavaScript">
window.captureEvents(Event.CLICK);
window.onclick= handle;
function handle(e) {
  document.links[1].handleEvent(e);
}
</script>
<a href="test.htm">"Кликните" по этой ссылке</a><br>
<a href="test.htm"
  onClick="alert('Обработчик событий для второй ссылки!');">Вторая ссылка</a>
</html>
```

Все сигналы о событиях Click, посылаются на обработку по второй ссылке - даже если Вы вовсе и не щелкнули ни по одной из ссылок!

Следующий скрипт демонстрирует, как Ваш скрипт может реагировать на сигналы о нажатии клавиш. Нажмите на какую-либо клавишу и посмотрите, как работает этот скрипт.

```
<html>
<script language="JavaScript">
window.captureEvents(Event.KEYPRESS);
window.onkeypress= pressed;
function pressed(e) {
  alert("Key pressed! ASCII-value: " + e.which); }
</script>
</html>
```

Вопросы для самоконтроля:

1 Обработка каких событий поддерживается в JavaScript 1.2?
2 Какие события отвечают за действия клавиатуры?
3 Для чего предназначена функция Resize?
4 Для чего предназначена функция Click?

Лабораторное занятие № 13

Тема: Drag & Drop

Цель: Изучить приемы создания документов, исользующих Drag & Drop

Задание:

1 Ознакомиться с теоретическим материалом по теме.

2 Создайте документ со следующим сценарием. Пользователь нажал клавишу мыши в каком-либо месте на окне браузера. Наш скрипт должен зафиксировать это событие и вычислить, с каким объектом (то есть слоем) это было связано.

3 Создайте документ, в котором демонстрируется применение MouseMove - текущие координаты курсора мыши отображаются в окне

состояния.

4 Объедините оба последних задания. Нужно, чтобы были представлены координаты указателя мыши, когда пользователь перемещает мышь, нажав на клавишу.

5 Создать документ, в котором нужно определить, по какому именно слою пользователь щелкнул клавишей мыши. И затем этот объект должен двигаться вслед за мышью.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

С помощью новой модели событий в языке JavaScript, 1.2 и механизма слоев можно реализовать на web-странице схему drag & drop ("перетащил и оставил"). Для этого понадобится, по крайней мере, Netscape Navigator 4.0, поскольку нужно пользоваться особенностями языка JavaScript 1.2.

Механизм drag & drop, который мы хотим здесь реализовать, ограничивается web-страницей. Поэтому нельзя использовать представленный здесь код, чтобы переносить объекты с HTML-страницы на жесткий диск вашего компьютера или другие подобные действия.

Язык JavaScript не поддерживает напрямую механизм drag & drop. Это значит, что нет возможности назначить объекту image свойство dragable (перемещаемый) или что-либо в этом роде. Поэтому мы должны сами писать необходимый для этого код. Это не так сложно. Для этого нужны две вещи. Во-вервых, нужнозарегистрировать определенные события, связанные с работой мышью, то есть нужно понять, каким образом, можно узнать, какой объект необходимо переместить и на какую позицию? Затем нужно подумать, каким именно образом будет показываться перемещение объектов по экрану. Конечно же, желательно пользоваться слоями при создании объектов и перемещении их по экрану. Каждый объект представлен собственным слоем.

События при работе с мышью в JavaScript 1.2

Какие события, происходящие при работе с мышью, следует использовать? Нет такого события, как*MouseDrag*, однако того же самого можно достичь, отслеживая события *MouseDown*, *MouseMove* и *MouseUp*. В версии 1.2языка JavaScript используется новая модель событий. И без нее нельзя решить эту задачу. Взглянем на некоторые важные ее части еще раз.

MouseDown, MouseMove и MouseUp

Вязыке JavaScript нетсобытия MouseDrag. Поэтому мы должны пользоваться событиями MouseDown, MouseMove и MouseUp, реализуя механизм drag & drop.

"Оставляемые" объекты

Предположим, нужно создать онлайновый магазин. Есть несколько изделий, которые можно поместить в корзину. Пользователь должен переносить эти изделия в корзинку и оставлять их там. Это означает, что нужно

регистрировать моменты, когда пользователь опускает некий объект в корзину - иными словами, что он хочет купить его.

Какую часть кода мы должны изменить, чтобы сделать такое? Нужно проверить, в какой месте оказался объект после того, как было зафиксировано событие *MouseUp* - то есть нужно сделать некоторые добавления к функции *endDrag()*. Например мы могли бы проверять, попадает ли в этот момент курсор мыши в границы некого прямоугольника. Если это так, то вызывается функция, регистрирующая все изделия, которые необходимо купить (например, можно поместить их в некий массив). Ну и после этого можное показывать это изделие уже в корзинке.

Реализации

Есть несколько путей для совершенствования скрипта. Во-первых, мы могли бы изменять порядок следования слоев, как только пользователь щелкает клавишей мыши по какому-либо объекту. Иначе выглядело бы странным, если бы вы перемещали объект, а он при этом прятался от вас за окружающие предметы. Очевидно, что эту проблему можно решить, меняя лишь порядок следования слоев в функции *startDrag()*.

Методические рекомендации к выполнению задания 2

Создайте документ со следующим сценарием. Пользователь нажал клавишу мыши в каком-либо месте на окне браузера. Наш скрипт должен зафиксировать это событие и вычислить, с каким объектом (то есть слоем) это было связано. Нам необходимо знать координаты точки, где произошло это событие. В JavaScript 1.2 реализован новый объект Event, который сохраняет координаты этой точки (а также еще и другую информацию о событии).

Другой важный момент заключается в перехвате событий. Если пользователь, например, щелкает по клавише мыши, то сигнал о соответствующем событии посылается непосредственно объекту button. Однако в нашем примере необходимо, чтобы событие обрабатывалось объектом window (окно). Поэтому мы позволяем объекту окна **перехватывать** сигнал о событии, связанном с мышью, т.е. чтобы именно объект window фиксировал это событие и имел возможность на него реагировать. Это демонстрируется в следующем примере (на примере события *Click*). Вы можете щелкнуть в любом месте окна браузера. При этом возникнет окно сообщения, где будут показаны координаты точки, где это событие имело место.

Код:

```
<html>
<script language="JavaScript">
<!--
  window.captureEvents(Event.CLICK);
  window.onclick= displayCoords;
  function displayCoords(e) {
    alert("x: " + e.pageX + " y: " + e.pageY);
}
// -->
```

```
</script>
"Кликните" клавишей мыши где-нибудь в этом окне.
</html>
```

Сперва мы сообщаем, что объект window перехватывает сигнал о событии *Click*. Для этого мы пользуемся методом *captureEvent()*. Строка

```
window.onclick= displayCoords;
```

говорит о том, что должно происходить, когда случается событие *Click*. Конкретнее, здесь сообщается, что в качестве реакции на событие *Click* браузер должен вызвать процедуру *displayCoords()* (Заметим, что Вам при этом **не следует**ставить скобки после слова *displayCoords*). В свою очередь, *displayCoords()* - это функция, которая определяется следующим образом:

```
function displayCoords(e) {
   alert("x: " + e.pageX + " y: " + e.pageY);
}
```

Как видите, эта функция имеет аргумент (мы назвали его *e*). На самом деле это объект Event, который передается на обработку функции *displayCoords()*. Объект Event имеет свойства *pageX* и *pageY* (наряду с другими), из которых моно получить координаты точки, где произошло событие. Окно с сообщением лишь показывает эти значения.

Методические рекомендации к выполнению задания 3

Создайте документ, в котором демонстрируется применение *MouseMove* - текущие координаты курсора мыши отображаются в окне состояния.

од скрипта почти такой же, как и в предыдущем задании:

```
<html>
<script language="JavaScript">
<!--
 window.captureEvents(Event.MOUSEMOVE);
 window.onmousemove= displayCoords;
 function displayCoords(e) {
  status= "x: " + e.pageX + " y: " + e.pageY;
}
// -->
</script>
```

Координаты мыши отображаются в строке состояния.

```
</html>
```

Заметьте, что нуно написать именно *Event.MOUSEMOVE*, где слово *MOUSEMOVE* обязательно должно быть написано заглавными буквами. А указывая, какая функция должна быть вызвана, когда произойдет событие MouseMove, Вы должны писать ее строчными буквами: *window.onmousemove=...*

Методические рекомендации к выполнению задания 4

Объедините оба последних задания. Нужно, чтобы были представлены координаты указателя мыши, когда пользователь **перемещает мышь, нажав**

на клавишу.

Код этого примера выглядит следующим образом:

```
<html>
<script language="JavaScript">
<!--
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;
function startDrag(e) {
  window.captureEvents(Event.MOUSEMOVE);}
function moveIt(e) {
  // показыватькоординаты
  status= "x: " + e.pageX + " y: " + e.pageY;}
function endDrag(e) {
  window.releaseEvents(Event.MOUSEMOVE);}
// -->
</script>
```

Нажмите на клавишу мыши и, не отпуская ее, передвиньте саму мышь. Координаты курсора будут отображаться в строке состояния.

```
</html>
```

Во-первых, мы заставляем объект window перехватывать сигналы о событиях *MouseDown* and *MouseUp*:

```
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
```

Как видно, мы пользуемся символом | (или), чтобы сказать, что объект window должен перехватывать несколько указанных событий. Следующие две строки описывают, что именно должно происходить, когда указанные события имеют место:

```
window.onmousedown= startDrag;
window.onmouseup= endDrag;
```

В следующей строке кода определяется, что происходит, когда объект window получает сигнал о событии *MouseMove*:

```
window.onmousemove= moveIt;
```

Однако постойте, мы же не определили *Event.MOUSEMOVE* в *window.captureEvents()*! Это означает, что данное событие не будет перехватываться объектом window. Тогда почему мы указываем объекту window вызывать *moveIt()*, раз сигнал об этом событии никогда не достигает объекта window? Ответ на этот вопрос можно найти в функции *startDrag()*, которая вызывается сразу после того, как произойдет событие *MouseDown*:

```
function startDrag(e) {
  window.captureEvents(Event.MOUSEMOVE);}
```

Это означает, что объект window начнет перехватывать событие *MouseMove*, как только будет нажата клавиша кнопка мыши. И мы должны прекратить перехватывать событие *MouseMove*, если произойдет событие *MouseUp*. Это делается в функции *endDrag()* с помощью метода *releaseEvents()*:

```
function endDrag(e) {
  window.releaseEvents(Event.MOUSEMOVE);}
```

Функция *moveIt()* записывает координаты мыши в окно состояния.

Теперь у нс есть все элементы скрипта, необходимые для регистрации событий, связанных с реализацией механизма drag & drop. И мы можем приступить к рисованию на экране наших объектов.

Методические рекомендации к выполнению задания 5

На предыдущих занятиях мы видели, как с помощью слоев можно создать перемещающиеся объекты. Все, что мы должны теперь сделать - это определить, по какому именно слою пользователь щелкнул клавишей мыши. И затем этот объект должен двигаться вслед за мышью.

Код задания:

```
<html>
<head>
<script language="JavaScript">
<!--
var dragObj= new Array();
var dx, dy;
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;
function startDrag(e) {
  currentObj= whichObj(e);
  window.captureEvents(Event.MOUSEMOVE);}
function moveIt(e) {
  if (currentObj != null) {
    dragObj[currentObj].left= e.pageX - dx;
    dragObj[currentObj].top= e.pageY - dy;  }}
function endDrag(e) {
  currentObj= null;
  window.releaseEvents(Event.MOUSEMOVE);}
function init() {
  // задать 'перемещаемые' слои
  dragObj[0]= document.layers["layer0"];
  dragObj[1]= document.layers["layer1"];
  dragObj[2]= document.layers["layer2"];}
function whichObj(e) {
  // определить, по какому объекту был произведен щелчок
var hit= null;
  for (var i= 0; i < dragObj.length; i++) {
    if ((dragObj[i].left < e.pageX) &&
        (dragObj[i].left + dragObj[i].clip.width > e.pageX) &&
        (dragObj[i].top < e.pageY) &&
```

```
        (dragObj[i].top + dragObj[i].clip.height > e.pageY)) {
          hit= i;
          dx= e.pageX- dragObj[i].left;
          dy= e.pageY- dragObj[i].top;
          break;    }
      }
       return hit;}
      // -->
      </script>
      </head>
      <body onLoad="init()">
      <layer name="layer0" left=100 top=200 clip="100,100" bgcolor="#0000ff">
      <font size=+1>Object 0</font>
      </layer>
      <layer name="layer1" left=300 top=200 clip="100,100" bgcolor="#00ff00">
      <font size=+1>Object 1</font>
      </layer>
      <layer name="layer2" left=500 top=200 clip="100,100" bgcolor="#ff0000">
      <font size=+1>Object 2</font>
      </layer>
      </body>
      </html>
```

Можно видеть, что в разделе <body> нашей HTML-страницы мы определяем три слоя. После того, как была загружена вся страница, при помощи программы обработки события *onLoad*, указанной в тэге <body>, вызывается функция *init()*:

```
function init() {
  // задать 'перемещаемые' слои
  dragObj[0]= document.layers["layer0"];
  dragObj[1]= document.layers["layer1"];
  dragObj[2]= document.layers["layer2"];}
```

Массив *dragObj* влючает все слои, которые пользователь может перемещать. Каждый такой слой получает в множестве *dragObj* некий номер. Его мы рассмотрим попозже.

Можно видеть, что мы используем тот же самый код, что использовался ранее для перехвата событий, связанных с мышью:

```
window.captureEvents(Event.MOUSEDOWN | Event.MOUSEUP);
window.onmousedown= startDrag;
window.onmouseup= endDrag;
window.onmousemove= moveIt;
```

К функции *startDrag()* я добавил следующую ср оку:

```
currentObj= whichObj(e);
```

Функция *whichObj()* определяет, по какому объекту был произведен щелчок. Возвращает она номер соответствующего слоя. Если ни один слой не был выделен, то возвращается значение *null*. Полученное значение хранится в

переменной *currentObj*. Это означает, что из *currentObj* можно извлечь номер слоя, который в данный момент необходимо перемещать (либо это будет *null*, если никакого слоя перемещать не надо).

В функции *whichObj()* для каждого слоя мы проверяем свойства *left*, *top*, *width* и *height*. По этим значеням мы и можем проверять, по которому из объектов пользователь щелкнул клавишей.

Вопросы для самоконтроля:

1 Что подразумевает технология drag & drop?

2 Поддерживает ли язык JavaScript напрямую механизм drag & drop?

3 Какие события, происходящие при работе мышью, существуют в JavaScript?

4 Каким образом осуществляется перехватка событий?

5 Охарактеризуйте событие MouseMove.

Лабораторное занятие № 14

Тема: Использование функций внутри формы

Цель: Рассмотреть принципы создания и применения функций внутри формы

Задание:

1 Ознакомиться с примеров скрипта.

2 Перепишите скрипт первого задания так, чтобы, открываясь, страница просила пользователя ввести имя. При выборе цвета должно всплывать окно со словами «Эй, (имя)! Вы выбрали (цвет)...».

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

Формы всегда начинаются командой <FORM> и заканчиваются командой </FORM>. Здесь ничего нового, простой HTML.

Рассмотрим пример:

```
<html>
<head>
<SCRIPTLANGUAGE="JavaScript">
function newcolor(color)
{  alert("Вывыбрали " + color)
  document.bgColor=color}
</SCRIPT>
</HEAD>
<BODY>
<p>Выбрать цвет фона</p>
<FORM>
```

```
 <INPUT TYPE="button" VALUE="Голубой"
    onClick="newcolor('lightblue')">
 <INPUT TYPE="button" VALUE="Розовый"
    onClick="newcolor('pink')">
<INPUT TYPE="button" VALUE="Вернуть"
    onClick="newcolor('white')">
</FORM>
</BODY>
</HTML>
```

Разбор скрипта:

Обратите внимание, мы передаем в функцию newcolor() (новый цвет) неизменяемую строку текста, стоящую в скобках ('lightblue'). Она находится в одинарных кавычках, потому что имя функции стоит в двойных.

Когда вы нажимаете кнопку, строка в скобках передается в функцию newcolor(). Функция ждет, пока поступит необходимая ей информация. Помните, во всех функциях до сих пор скобки были пустые? Потому что у них были все необходимые данные. В данном случае дополнительная информация поступает в функцию, когда пользователь нажимает на кнопку. Кнопка содержит ту же функцию, только теперь у нее есть необходимые данные, то есть цвет.

Форма передает цвет двум элементам в разделе <SCRIPT>: методу alert и строке document.bgColor. Получив все данные, функция вступает в действие: всплывает окно и меняется цвет фона. Не запутайтесь: VALUE (значение) в команде INPUT не является свойством JavaScript, она помещает текст на кнопку.

Методические рекомендации к выполнению задания 2

Перепишите скрипт так, чтобы, открываясь, страница просила пользователя ввести имя. При выборе цвета должно всплывать окно со словами «Эй, (имя)! Вы выбрали (цвет)...».

Примерный вид решения:

Чтобы добиться нужного эффекта, вставьте запрос перед функцией, а потом результат запроса в команду alert:

```
<html>
<head>
<SCRIPT LANGUAGE="JavaScript">
varuser_name = prompt ("Можно узнать, как Вас зовут?","Ваше имя");
function newcolor(color)
{alert("Эй, " + user_name + "! Вывыбрали " + color)
document.bgColor=color}
</SCRIPT>
</head>
<body bgcolor="xxxxxx">
<center><h3>Фон страницы</h3>
</center>
```

```
<form>
  <input type="button" value="Оранжевый" onClick="newcolor('orange')">
  <input type="button" value="Салатовый"
onClick="newcolor('lightgreen')">
</form>
</body>
</html>
```

Вопросы для самоконтроля:

1 Для чего предназначены формы?
2 Укажите особенности использования функций в формах.
3 Какую роль выполняет функция newcolor()?

Лабораторное занятие № 15

Тема: Передача информации в функцию

Цель: Рассмотреть принципы передачи информации

Задание:

1 Рассмотрите приведенный пример скрипта.

2 Составьте документ HTML с формой **aform**. В ней должно быть два текстовых поля, одно для геометрической фигуры, другое для цвета, и кнопка. Напишите функцию с переменной, которая содержит слова «Мне нравится ». Когда пользователь нажмет на кнопку, должно всплывать окно со следующей надписью:

– Мне нравится **геометрическая фигура такого-то цвета**. (по результатам тех данных, которые пользователь вводит в форму)

– Покажите длину (length) «фигуры».

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

Рассмотримпример:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function doit()
{  alert("document.myform.fname.value — это  "
+ document.myform.fname.value)
  var greeting="Привет, "
  alert(greeting + document.myform.fname.value
+ " " + document.myform.lname.value)
  alert("Количествобуквввимени "
```

```
+ document.myform.fname.value.length)}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform">
  Вашеимя:
     <INPUT TYPE="text" NAME="fname"><p>
  Вашафамилия:
     <INPUT TYPE="text" NAME="lname"><p>
     <INPUT TYPE="button" VALUE="Отправить" onClick="doit()">
</FORM>
</BODY>
</HTML>
```

Разбор скрипта

Начнем с элементов формы:

```
<FORM NAME="myform">
  Вашеимя:
     <INPUT TYPE="text" NAME="fname"><p>
  Вашафамилия:
     <INPUT TYPE="text" NAME="lname"><p>
  <INPUT TYPE="button" VALUE="Отправить" onClick="doit()">
  </FORM>
```

Видите, мы дали форме имя **myform**. Поле ввода для имени пользователя названо **fname**, а поле для фамилии — **lname**. Теперь у каждого элемента есть имя.Данные, которые введет пользователь, станут значением (value) соответствующих текстовых полей. Понятно? Тому, что написано в поле **fname**, будет присвоено имя **fname**.Когда пользователь нажмет на кнопку (обработчик события onClick), запустится функция doit().

Теперь посмотрим на функцию:

```
functiondoit()
{   alert("document.myform.fname.value — это  "
+ document.myform.fname.value)
  var greeting="Привет, "
  alert(greeting + document.myform.fname.value + " "
+ document.myform.lname.value)
 alert("Количествобукввимени "
+ document.myform.fname.value.length) }
```

Такой передачи данных, как на предыдущих уроках, не происходит. Видите, в скобках функции doit() ничего нет. Но по иерархическим командам понятно, что функция вызывает данные, введенные в форму.

Мы тщательно следуем иерархии объектов: за объектом документ следует объект форма (на него указывает имя формы, **myform**), за ним следует объект поле формы (на него указывает имя поля, **fname**), за ним следует свойство значение (**value**). Без свойства **value** данные, переданные пользователем, не попали бы в иерархическую команду.

Дальше переменная **greeting** (приветствие). Приветствие показано в команде **alert(greeting)**.

Когда пользователь нажимает на кнопку, всплывает окно с его именем.

Второе окно включает в себя переменную greeting. Появляется надпись: «Привет, (имя) (фамилия)», составленная с помощью данных, полученных через форму. Еще раз обратите внимание на **value.**

Наконец всплывает третье окно с неким текстом и вызывает следующее: **document.myform.fname.value.length**. Это команда, которая передает длину (length) слова, введенного в поле формы. Если fname содержит «Коля», то длина равна 4. Команда length следует за value.Таким образом она точно сосчитает буквы в тексте, а не что-нибудь другое. length — это тоже свойство.

Методические рекомендации к выполнению задания 2

Составьте документ HTML с формой **aform**. В ней должно быть два текстовых поля, одно для геометрической фигуры, другое для цвета, и кнопка. Напишите функцию с переменной, которая содержит слова «Мне нравится ». Когда пользователь нажмет на кнопку, должно всплывать окно со следующей надписью:

1. Мне нравится **геометрическая фигура такого-то цвета.** (по результатам тех данных, которые пользователь вводит в форму)
2. Покажите длину (length) «фигуры».

Пример решения:

```
<script language="javascript">
function say()
{var first = document.aform.first.value;
var second = document.aform.second.value;
var like = "Мненравится "
alert(like + first + " " + second)
alert("В названии вашей любимой геометрической фигуры " +
document.aform.second.value.length + " букв.")}
</script></head><body>
<form name="aform">
Вашлюбимыйцвет: <br>
<input type="text" name="first" size="20"><br>
Ваша любимая геометрическая фигура: <br>
<input type="text" name="second" size="20"></p>
<p><input type="button" value="Отослать" onClick="say()"></p>
</form>
</body>
</html>
```

Нужно было заменить имена полей на first и second. Потом убрать первую команду alert из прошлого урока, а все остальное почти без изменений.

Вопросы для самоконтроля:

1 Каким образом осуществляется передача информации в функцию?

2 В каких случаях используется передача информации в функцию?

Лабораторное занятие № 16

Тема: Условный оператор

Цель: Рассмотреть принципы создания веб-страниц с использованием условного оператора

Задание:

1 Ознакомиться с примером скрипта с использованием условного оператора.

2 Перепишите пример из задания 1 так, чтобы он спрашивал, какого вы пола. Пусть в зависимости от ответа меняется фоновый цвет страницы. Помните, что в JavaScript различаются строчные и заглавные буквы, так что будьте внимательны в своих условиях.

3 Измените скрипт второго примера так, чтобы при неверной догадке он сообщал пользователю, что он назвал слишком большое или слишком маленькое число.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1

За **IF** следует условие и указание, что делать, если оно верно. Верным может быть одно условие или несколько. Скрипт знает, где начинаются и кончаются верные условия, потому что они заключены в {фигурные скобки}.

Разберемпример:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function askuser() {
var answer=" "
varstatement="Отвечай, даилинет"
var answer=prompt("Любишьгорчицу?")
if ( answer == "да")
{statement="Я тоже обожаю горчицу!"}
if(answer == "нет")
{statement="Я тоже горчицу терпеть не могу!"}
alert(statement) }
</SCRIPT>
</HEAD>
<BODY>
<h2>Горчица</h2>
<FORM>
```

<INPUT TYPE="button" VALUE="Жми!" onClick="askuser()">
</FORM>
</BODY>
</HTML>

Разборскрипта:

Начнем с кнопки:

<FORM>
<INPUT TYPE="button" VALUE="Жми!" onClick="askuser()">
</FORM>

Здесь ничего нового, простая форма с кнопкой, которая запускает функцию askuser() (спросить пользователя).

Фрагмент скрипта с функцией:

functionaskuser() {
var answer=" "
varstatement="Отвечай, даилинет"
var answer=prompt("Любишьгорчицу?")
if (answer == "да")
{statement="Я тоже обожаю горчицу!"}
if(answer == "нет")
{statement="Я тоже горчицу терпеть не могу!"}
alert(statement)}

Значение переменной **answer** (ответ) равно тому, что введет пользователь по запросу.

На все те случаи, когда пользователь не отвечает «да» или «нет», создается переменная **statement** (заявление).

Дальше запрос prompt приравнивается к **answer**. Теперь у нас две переменные под одним именем. Пока имейте это в виду.

Следом за **if** идет условие в (круглых скобках).

В условии ставим не один, а **два знака равенства ==**! Одинарный знак = используется вне скобок.

Помните, отрезки текста ставятся в кавычки.

Вот как разворачивается действие:

- Запрашивается ваше мнение;
- Скрипт сверяет его с условиями;
- Если ответ «да», появляется окно со словами: «Я тоже обожаю горчицу!»
- Если ответ «нет», появляется окно со словами: «Я тоже терпеть не могу горчицу!»
- Если ответ ни тот, ни другой, тогда переменная answer остается пустой и заявление «Отвечай, да или нет» отсылается в alert.

Помните, что JavaScript чувствителен к регистру. То есть если вы напишете **«НЕТ»** или **«Нет»**, условие не будет выполнено! Чтобы условие было верно, необходимо ввести **«нет»**. Исправить это можно, добавив еще несколько условий IF на все случаи жизни.

If/Else (если/иначе) дают вам дополнительный контроль над

программой, позволяя принимать решения на оба случая: и когда условие выполнено, и когда не выполнено.

Рассмотримпример:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function rand()
  {now=new Date()
   num=(now.getSeconds())%10
   num=num+1   }
function guessnum()
  {guess=prompt("Угадай, какое?")
   if (eval(guess) == num)
   {alert("ПРАВИЛЬНО!!!")
   rand()    }
    Else
   alert("Нет. Попробуй еще раз.")      }
 </SCRIPT>
<BODYonLoad="rand()">
 <h3>Я загадал число от 1 до 10</h3>
 <FORM NAME="myform">
    <INPUT TYPE="button" VALUE="Угадай"
      NAME="b1" onClick="guessnum()">
  </FORM>
</BODY>
</HTML>
```

Разберем скрипт:

Начнем со строки BODY:

```
<body bgcolor="xxxxxx" onLoad="rand()">
```

На этот раз функция запускается не кнопкой через onClick, а командой onLoad, чтобы к тому времени, когда пользователь нажмет на кнопку, число уже было выбрано. Если сделать это, как в прошлом уроке, то каждый раз, нажимая на кнопку, вы будете получать новое число. А оно должно оставаться одним и тем же, пока вы гадаете.

Перваяфункция:

```
function rand() {
now=new Date()
num=(now.getSeconds())%10
num=num+1}
```

Функция выбирает наугад число от 0 до 9 и привязывает его к **num.** Потом прибавляет к num единицу, чтобы выбор осуществлялся между 1 и 10.

Вторая функция:

```
functionguessnum()
  {guess=prompt("Угадай, какое?")
  if (eval(guess) == num)
```

```
{alert("ПРАВИЛЬНО!!!")
rand()    }
 Else
alert("Нет. Попробуй еще раз.")
 }
```

Этот фрагмент написала Эндри, потому что она очень умная. В памяти компьютера уже есть число, полученное через первую функцию. Вторая дает вам возможность угадать его. Смотрите, что происходит:

- С помощью запроса создается переменная **guess** (догадка). Функция **eval()** вычисляет или выполняет строку в скобках (выражение, команду или последовательность команд) и подставляет полученное значение вместо себя. Она не является методом какого-либо объекта, но может использовать свойства уже существующего. В данном случае переводит текст, полученный с помощью запроса, в нашу функцию для последующей обработки. Обратите внимание на {фигурные скобки}.
- Переходим к IF/Else. Если (if) guess (догадка) равна загаданному числу num, тогда запускается команда alert("ПРАВИЛЬНО").
- Если это не так, а иначе (else), тогда запускается другая команда alert.

Это вам уже знакомо:

```
<formname="myform">
<input type="button" value="Угадай" name="b1" onClick="guessnum()">
</form>
```

Кнопка запускает функцию, которая дает возможность угадать задуманное число.

Методические рекомендации к выполнению задания 2

Перепишите пример 1 так, чтобы он спрашивал, какого вы пола. Пусть в зависимости от ответа меняется фоновый цвет страницы. Помните, что в JavaScript различаются строчные и заглавные буквы, так что будьте внимательны в своих условиях.

Возможно, это задание оказалось не таким уж простым, но я надеюсь, что вы с ним справились. Нужно было кое-где поменять текст, но главное - заменить команду alert на document.bgColor.

```
<script language="javascript">
function askuser() {
varanswer=prompt("Вы мужчина или женщина?")
if ( answer == "женщина")
{document.bgColor="FE92B5"}
if(answer == "мужчина")
{document.bgColor="70E4F1"}}
</script>
<form>
<input type="button" value="Поговоримосексе" onClick="askuser()">
<form>
```

Методические рекомендации к выполнению задания 3

Измените скрипт второго примера так, чтобы при неверной догадке он сообщал пользователю, что он назвал слишком большое или слишком маленькое число.

В этом случае возможны только три решения: слишком много, слишком мало или правильно. Подумайте вот о чем: нужна ли вам команда Else или сойдет и парочка дополнительных If?

Возможное решение:

Возможны только три результата: слишком мало, слишком много и точно. То есть каждый раз, когда пользователь вводит свою догадку, будет задействовано одно из трех условий. Здесь вам даже не понадобится ELSE. Таким образом, требуется только три утверждения IF.

Обратите внимание на команды < и > в скрипте. В данном случае они означают то же самое, что и на уроках математики: больше и меньше.

Скрипт

```
<html>
<head>
  <script language="JavaScript">
    function rand()
      {now=new Date()
       num=(now.getSeconds())%10
       num=num+1      }
    function guessnum()
      {guess=prompt("Угадай, какое?")
      if (eval(guess) == num)
        {alert("Точно!!!")        }
      if(eval(guess) > num)
       {alert("Слишкоммного, жмиеще.")}
      if(eval(guess) < num)
       {alert("Слишкоммало, жмиеще.")}     }
  </script>
<body bgcolor="white" onLoad="rand()">
 <h2>Я загадал число от 1 до 10</h2>
 <form name="myform">
    <input type="button" value="Угадай" name="b1" onClick="guessnum()">
  </form>
</body>
</html>
```

Вопросы для самоконтроля:

1 В каких случаях используется условный оператор?
2 Для чего предназначен условный оператор?
3 Каким образом производится организация условного оператора?

Лабораторное занятие № 17
Тема: Работа с массивами
Цель: Рассмотреть принципы организации массивов на веб-страницах

Задание:
1 Рассмотреть пример программы с использованием массива.
2 Напишите программу JavaScript, которая содержит кнопку с надписью: «Щелкните, чтобы попасть на случайный сайт». Когда пользователь нажмет на нее, запустится функция, которая наугад выберет число и сайт из массива внутри команды JavaScript top.location.href = urls[num]. top (вершина) — это свойство объекта window, оно относится к главному окну браузера. location.href, другой объект со свойством, содержит адрес URL.

Необходимые приборы: ПК, текстовый редактор Блокнот, браузер

Методические рекомендации к выполнению лабораторной работы:

Методические рекомендации к выполнению задания 1
Каждая **переменная** имеет одно значение, но иногда вам необходим **массив (array)**, или переменная, которая имеет множество значений.
Рассмотрим пример, в котором программа просит пользователя угадать телевизионный канал из перечня телеканалов. Запрос повторяется до тех пор, пока пользователь не угадает. Каждый раз при нажатии кнопки выбирается новый телеканал.
Скрипт:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
 tv=new Array()
tv[0]="ОРТ"
tv[1]="РТР"
tv[2]="ТВЦ"
tv[3]="НТВ"
tv[4]="ТВ6"
num=0
function picktv()
{now=new Date()
num=(now.getSeconds())%5}
function whichtv(){
picktv()
guess=" "
while (tv[num] != guess.toUpperCase())
{guess=prompt("Угадайте мой любимый телеканал:
ОРТ, РТР, ТВЦ, НТВ или ТВ6?")
```

```
if (guess.toUpperCase() == tv[num])
{alert("Это мой любимый телеканал!")}
 Else
 {alert("Нет, попробуйте еще раз.")}} }
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Угадайте"
onClick="whichtv()">
</FORM>
</BODY>
</HTML>
```

Разборскрипта

```
<SCRIPT LANGUAGE="JavaScript">
 tv=new Array()
 tv[0]="ОРТ"
 tv[1]="РТР"
 tv[2]="ТВЦ"
 tv[3]="НТВ"
 tv[4]="ТВ6"
 num=0
```

- **tv=newArray()** объявляет, что tv представляет собой массив. С пустыми **(скобками)** массив может быть какой угодно длины. Можно также указать длину массива, например, **tv=new Array(5).**
- Помните, что массив array может иметь множество значений. Можно представить себе массив в виде таблицы:

tv[0]	tv[1]	tv[2]	tv[3]	tv[4]
ОРТ	РТР	ТВЦ	НТВ	ТВ6

- Обратите внимание, мы заранее указываем переменную **num**, у которой одно значение, равное 0, и массив **tv**, который имеет 5 значений.
- Теперь функция **picktv()**:

```
function picktv()
   {       now=new Date()
     num=(now.getSeconds())%5}
```

- Функция **picktv()** наугад выбирает число от 0 до 4, которое становится индексом **tv**. Помните, от нуля до четырех ПЯТЬ чисел. То есть если **num** равно 2, то любимый телеканал — **tv[2],** или **ТВЦ.**
- Теперь функции **whichtv()** и **picktv()**:

```
function whichtv()
{
picktv()
guess=" "
while (tv[num] != guess.toUpperCase())
{guess=prompt("Угадайте мой любимый телеканал:
```

```
 ОРТ, РТР, ТВЦ, НТВ или ТВ6?")
 if (guess.toUpperCase() == tv[num])
  {alert("Это мой любимый телеканал!")}
  Else
  {alert("Нет, попробуйте еще раз.")}}
  }
```

Команда guess=prompt должна находиться полностью на одной строке.

- Вот кое-что новенькое! Видите, первым делом функция вызывает другую функцию, **picktv()**. Таким образом, когда бы вы ни нажали на кнопку, будет выбираться новый телеканал.
- Строка**while (tv[num] != guess.toUpperCase())**. **Метод** или действие **toUpperCase()** (в верхний регистр) используется для перевода всего, что бы вы ни напечатали, в верхний регистр.
- Программа повторяет цикл **While**, пока пользователь не угадает правильный телеканал. Фрагмент с циклом While уже должен казаться вам вполне знакомым.
- Обратите внимание на **If** и **Else**. В игре возможны только два результата: либо вы правы, либо ошибаетесь.
- Теперь кнопка, которая все это запускает:

```
<FORM>
<INPUT TYPE="button" VALUE="Угадай"
  onClick="whichtv()">
</FORM>
```

Еще кое-что о массивах:

В JavaScript есть несколько встроенных массивов. В массивах можно указывать формы. Можете передать форму командой **document.myform** или **document.forms[0]**, если это первая форма. Массивы всегда начинаются с нуля. Вторая будет **document.forms[1]**. Третья **document.forms[2]** и так далее...

Для рисунков тоже есть готовый массив. Можно указать **pic1.gif** как **document.pic1.src** или как **document.images[0].src**. Просто продолжайте следовать схеме, прибавляя номер в [квадратных скобках].

Методические рекомендации к выполнению задания 2

Напишите программу JavaScript, которая содержит кнопку с надписью: «Щелкните, чтобы попасть на случайный сайт». Когда пользователь нажмет на нее, запустится функция, которая наугад выберет число и сайт из массива внутри команды JavaScript top.location.href = urls[num]. top (вершина) — это свойство объекта window, оно относится к главному окну браузера. location.href, другой объект со свойством, содержит адрес URL.

Примерныйскрипт:

```
<html>
<head>
<script language="JavaScript">
 url=new Array()
 url[0]="http://www.jsp.newmail.ru/les5.htm"
```

```
url[1]= "http://www.jsp.newmail.ru/les10.htm"
url[2]= "http://www.jsp.newmail.ru/les15.htm"
 url[3]="http://www.jsp.newmail.ru/les20.htm"
function rand()
 { now=new Date()
 num=(now.getSeconds())%4
 top.location.href = url[num] }
</script>
</head>
<body>
<center>
<h2>Случайнаяурла</h2>
<form>
<input type="button" value="Зашлемваснеизвестнокуда!"
onClick="rand()">
</form>
</center>
</body>
</html>
```

Вопросы для самоконтроля:

1 Каким образом осуществляется работа с массивами?
2 В каких случаях обязательно использование массивов?
3 Дайте определение понятия массив.
4 Какие виды массивов существуют?
5 Какие команды используются для организации работы с массивами?