1. Первые шаги

1.1. Что такое JavaScript

JavaScript - новый язык для составления скриптов, разработанный фирмой Netscape
Код скрипта JavaScript размещется непосредственно на HTML-странице. Чтобы увидеть, как это делается, давайте рассмотрим следующий простой пример:

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

Все, что стоит между тэгами <script> и </script>, интерпретируется как код на языке JavaScript. Здесь Вы также видите пример использования инструкции *document.write()* - одной 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). Так наша небольшая программа на JavaScript в HTML-документе пишет фразу *"А это JavaScript!"*.

1.2. Основы программирования на Java Script. Управление потоком вычислений

Вообще, все типы операторов, которые поддерживаются обычными языками программирования, реализованы JavaScript (+,-,*, /, %, >>,<<, +=, -=, ...). При этом оператор сложения "+" при работе со строками означает конкатенацию последних, т.е. добавление в конец строки новую строку:

```
s = "string1"+"string2"
```

Кроме операций с числами и описаний стандартных классов в JavaScript есть команды управления потоком вычислений:

- *for* - цикл;

```
for(i=0;i<9;i++)
{
...
}
```

- *for* - цикл свойств объекта (переменных определенных в классе);

```
for(i in obj)
{
 str = obj[i]
}
```

- *if..else* - условный оператор;

```
if(i>0)
{
 ...
}
else
{
 ...
}
```

- *while* - условный цикл;

```
while(j==k)
{
 j++;
 k--;
}
```

- *break* - принудительный выход из цикла;

```
while(i<6)
{
 if(i==3) break;
}
```

- *continue* - переход в конец цикла;

```
while(i &lt 6)
{
 if(i==3) continue;
}
```

- *var* - оператор объявления переменной.

```
var kuku = "kuku"
```

Тип переменной определяется по присвоенному ей значению.

1.3. События

События и обработчики событий являются очень важной частью для программирования на языке JavaScript. События, главным образом, инициируются теми или иными действиями пользователя:

- *onLoad* - выполнение скрипта или функции при загрузке;
- *onChange* - порождается при изменении значения элемента формы;
- *onClick* - порождается при выборе объекта (button, checkbox и т.п.);
- *onSelect* - порождается при выборе текстового объекта (text, textarea);
- *onSubmit* - при нажатии на кнопку Submit;
- *onUnload* - при переходе к другой странице.

Мы можем заставить нашу JavaScript-программу реагировать на некоторые из них. И это может быть выполнено с помощью специальных программ обработки событий.

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

Функция *alert()* позволяет Вам создавать выпадающие окна. При ее вызове Вы должны в скобках задать некую строку. В данном примере мы использовали и двойные, и одинарные кавычки. Если бы мы написали `onClick="alert("Yo")"`, то компьютер не смог бы разобраться в нашем скрипте.

Вы можете использовать в скрипте множество различных типов функций обработки событий.

1.4. Массивы

Первый тип новых объектов, которые мы рассмотрим, являются массивы. Тип "Array" введен в JavaScript 1.1 для возможности манипулирования самыми разными объектами

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

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

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

В данном случае массив будет состоять из 6 элементов, т.к. первым элементом массива считается элемент с индексом 0. Для массивов определены три метода: `join, reverse,`

sort. Join объединяет элементы массива в строку символов, в качестве аргумента в этом методе задается разделитель:

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

В результате выполнения присваивания значения строке символов string мы получим следующую строку:

```
string = "red + white + blue"
```

Другой метод, reverse, изменяет порядок элементов массива на обратный, а метод sort отсортировывает их в порядке возрастания. У массивов есть два свойства: length и prototype. Length определяет число элементов массива. Если нужно выполнить некоторую рутинную операцию над всеми элементами массива, то можно воспользоваться циклом типа:

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

Свойство prototype позволяет добавить свойства к объектам массива. Однако наиболее часто, в программе на JavaScript используются встроенные массивы, главным образом графические образы (Images) и гипертекстовые ссылки (Links).

1.5. Функции

В большинстве наших программ на языке JavaScript мы будем пользоваться функциями.

```
<html>
<script language="JavaScript">
<!-- hide
function myFunction() {
  document.write("Это JavaScript!<br>");
}
myFunction();
myFunction();
myFunction();
// -->
</script>
</html>
```

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.

Почему собственно функции столь важны в JavaScript? Функции обычно используются совместно с процедурами обработки событий. Рассмотрим следующий пример:

```
<html>
<head>
<script language="JavaScript">
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>
```

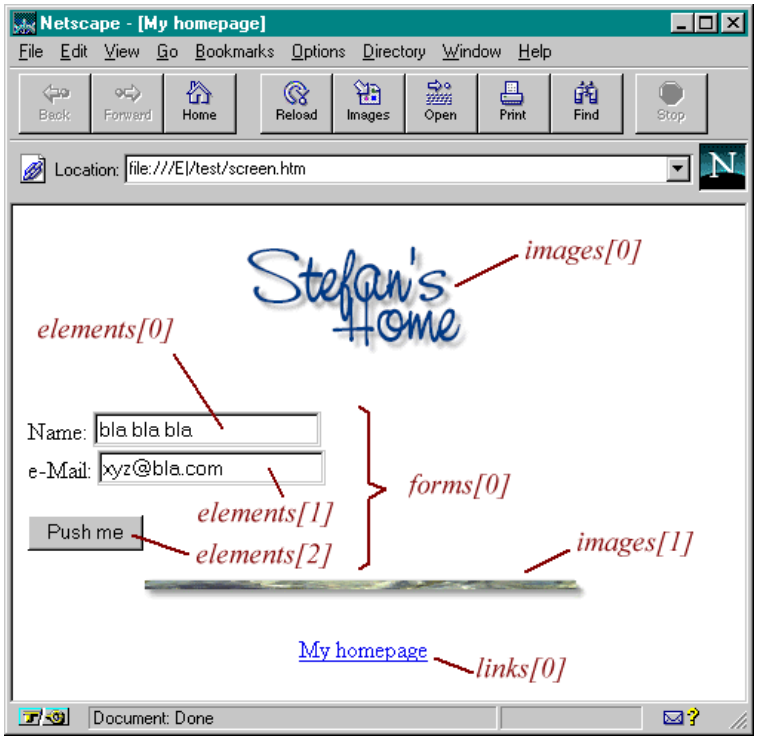
Здесь при нажатии на кнопку осуществляется вызов функции *calculation()*.Иными словами, мы получаем выпадающее окно, в котором написано число 17.

2. Документ HTML.

2.1. Иерархия объектов в JavaScript

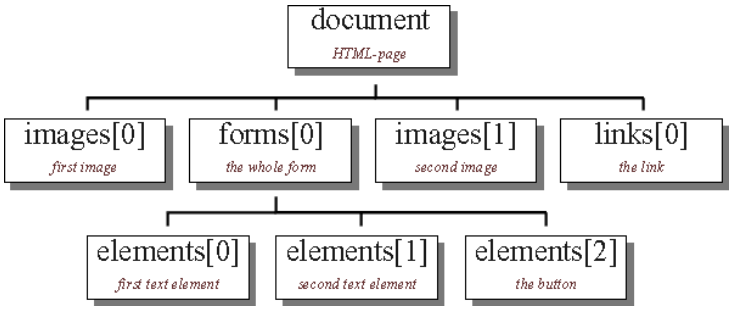
В языке JavaScript все элементы на web-странице выстраиваются в иерархическую структуру. Каждый элемент предстает в виде объекта. В свою очередь, язык JavaScript позволит Вам легко управлять объектами web-страницы, хотя для этого очень важно понимать иерархию объектов, на которые опирается разметка HTML. Как это все действует, Вы сможете быстро понять на следующем примере. Рассмотрим простую HTML-страницу:

```
<html>
<head>
<title>My homepage
</head>
<body bgcolor=#ffffff>
<center>
<img src="home.gif" name="pic1">
</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="Push
me" name="myButton"
onClick="alert('Yo')">
</form>
<p>
<center>
<img src="ruler.gif" name="pic4">
<p>
<a href="http://yyyy.de/~qqq/">My
homepage</a>
</center>
</body>
</html>
```



А вот как выглядит эта страница на экране (я добавил к ней еще красным цветом комментарии):

Итак, мы имеем два рисунка, одну ссылку и некую форму с двумя полями для ввода текста и одной кнопкой. С точки зрения языка JavaScript окно браузера - это некий объект *window*, внутри которого размещается документ HTML. Такая страница является ни чем иным, как объектом *document*. Объект document является очень важным объектом в языке JavaScript т.к. все без исключения объекты HTML являются свойствами объекта document. На следующем рисунке иллюстрируется иерархия объектов, создаемая HTML-страницей из нашего примера:



Разумеется, мы должны иметь возможность получать информацию о различных объектах в этой иерархии и управлять ею. Для этого мы должны знать, как в языке JavaScript организован доступ к различным объектам. 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]*. Доступ к первому полю для ввода текста можно получить, записав:

document.forms[0].elements[0]. Элемент, соответствующий полю для ввода текста, имеет свойство *value*, которое как раз и соответствует введенному тексту.

Если Вы имеете дело с большими страницами, то процедура адресации к различным объектам по номеру может стать весьма запутанной. Во избежание подобной проблемы, Вы можете сами присваивать различным объектам уникальные имена.

```
<form name="myForm">
Name:
<input type="text" name="name" value=""><br>
...
```

Эта запись означает, что объект *forms[0]* получает теперь еще и второе имя - *myForm*. Таким образом, вместо

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

Вы можете записать

```
name= document.myForm.name.value;
```

2.2. Объект location

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.

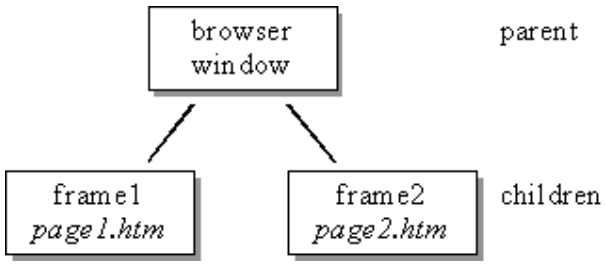
Впрочем, для нас гораздо более важно, что Вы имеете возможность записывать в *location.href* свои новые значения. Например, в данном примере кнопка загружает в текущее окно новую страницу:

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

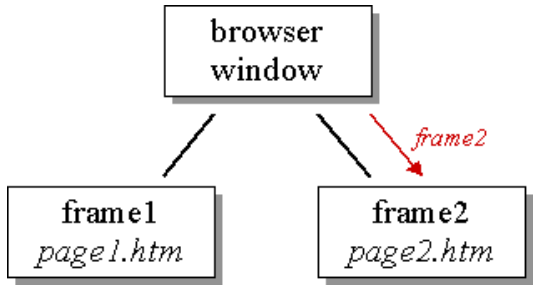
3. Фреймы

3.1. Фреймы и JavaScript

А теперь давайте посмотрим, как JavaScript "видит" фреймы, присутствующие в окне браузера. Для этой цели мы создадим два фрейма, как было показано в первом примере этой части описания.



Как мы уже видели, JavaScript организует все элементы, представленные на web-странице, в виде некой иерархической структуры. То же самое относится и к фреймам. 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 мы можем обмениваться информацией с двумя указанными фреймами.

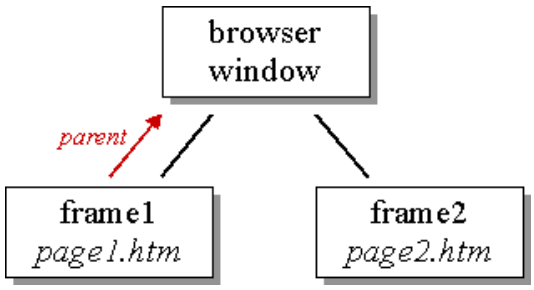


Если Вы пишете скрипт для родительского окна - то есть для страницы, создающей эти фреймы - то можете обращаться к этим фреймам, просто называя их по имени. Например, можно написать:

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

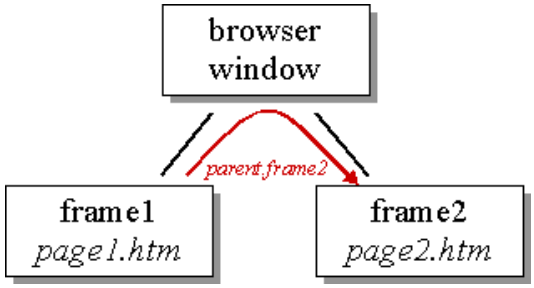
В некоторых случаях Вам понадобится, находясь во фрейме, получать доступу к родительскому окну. Например, это бывает необходимо, если

Вы хотите 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 мы хотим 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, мы можем 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. Мы не можем просто так вызвать *frame2*, находясь в фрейме *frame1*, который просто ничего не знает о существовании второго фрейма. 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.");
```

3.2. Навигационные панели

Давайте рассмотрим, как создаются навигационные панели. В одном фрейме мы создаем несколько ссылок. Однако, если посетитель активирует какую-либо из них, соответствующая страница будет помещена не в тот же самый фрейм, а в соседний.

Пример этого:

Сперва нам необходимо написать скрипт, создающий указанные фреймы:

frames3.htm

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

Следующая web-страница будет загружена во фрейм "*menu*":

menu.htm

```
<html>
<head>
<script language="JavaScript">
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>
```

Здесь Вы можете увидеть несколько способов загрузки новой страницы во фрейм *main*. В первой ссылке для этой цели используется функция *load()*. Во второй ссылке присутствует параметр *target*. На самом деле это уже не относится к JavaScript. Это одна из конструкций языка HTML. Как видно, мы всего лишь указываем имя необходимого фрейма. И на примере третьей ссылки Вы можете видеть, как с помощью *target* можно избавиться от фреймов.

А если Вы хотите избавиться от фреймов с помощью функции *load()*, то Вам необходимо написать в ней лишь `parent.location.href= url`.

4. Формы

4.1. Проверка информации, введенной в форму

Допустим, HTML-страница содержит два элемента для ввода текста. В первый из них пользователь должен вписать свое имя, во второй элемент - адрес для электронной почты. Что касается информации, введенной в первый элемент, то Вы будете получать сообщение об ошибке, если туда ничего не было введено. Любая представленная в элементе информация будет рассматриваться на предмет корректности. Конечно, это не гарантирует, что пользователь введет не то имя. Браузер даже не будет возражать против чисел. Например, если Вы ввведете '17', то получите приглашение 'Hi 17!'. Так что эта проверка не может быть идеальна.

Второй элемент формы несколько более сложнее. Признаком того, что пользователь правильно ввел адрес электронной почты служит наличие символа @.

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

```
<html>
<head>
<script language="JavaScript">
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('@') == -1)
        alert("Неверно введен адрес e-mail!");
  else alert("OK!");
}
</script>
</head>
<body>
<form name="first">
Введите Ваше имя:<br>
<input type="text" name="text1">
<input type="button" name="button1" value="Проверка" onClick="test1(this.form)">
<P>
Введите Ваш адрес e-mail:<br>
<input type="text" name="text2">
<input type="button" name="button2" value="Проверка" onClick="test2(this.form)">
</body>
</html>
```

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'

Рассмотрим теперь функцию *test2(form)*. Здесь вновь сравнивается введенная строка с пустой - "" и проверяется наличие символа @.

4.2. Проверка на присутствие определенных символов

В некоторых случаях Вам понадобится ограничивать информацию, вводимую в форму, лишь некоторым набором символов или чисел. Достаточно вспомнить о телефонных. Решение задачи продемонстрировано в следующем примере:

Исходный код этого скрипта:

```
<html>
<head>
<script language="JavaScript">
function check(input) {
  var ok = true;

  for (var i = 0; i < input.length; i++) {
    var chr = input.charAt(i);
    var found = false;
    for (var j = 1; j < check.length; j++) {
      if (chr == check[j]) found = true;
    }
    if (!found) ok = false;
  }
  return ok;
}
function test(input) {
  if (!check(input, "1","2","3","4","5","6","7","8","9","0") {
    alert("Input not ok.");
  }
  else {
    alert("Input ok!");
  }
}
</script>
</head>
<body>
<form>
Telephone:
<input type="text" name="telephone" value="">
<input type="button" value="Check"
  onClick="test(this.form.telephone.value)">
</form>
</body>
</html>
```

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

4.3. Предоставление информации, введенной в форму

Какие существуют возможности для передачи информации, внесенной в форму? Самый простой способ состоит в передаче данных формы по электронной почте.

Соответствующий скрипт будет простым текстом HTML. Я должен лишь добавить, что команда mailto работает не повсюду - например, поддержка для ее отсутствует в Microsoft Internet Explorer 3.0.

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

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

```
  choice=3
  submit=Send.
```

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

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

...
```

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

5. Окна и динамически создаваемые документы

5.1. Создание окон

Открытие новых окон в браузере - грандиозная возможность языка JavaScript. Вы можете либо загружать в новое окно новые документы (например, те же документы HTML), либо (динамически) *создавать* новые материалы.

Заметим, что Вы имеете возможность управлять самим процессом создания окна. Например, Вы можете указать, должно ли новое окно иметь строку статуса, панель инструментов или меню. Например, в следующем скрипте открывается новое окно размером 400x300 пикселов, которое не имеет ни строки статуса, ни панели инструментов, ни меню.

```
<html>
<head>
<script language="JavaScript">
function openWin2() {
  myWin= open("bla.htm", "displayWindow",
    "width=400,height=300,status=no,toolbar=no,menubar=no");
}
</script>
</head>
<body>
<form>
<input type="button" value="Открыть новое окно" onClick="openWin2()">
</form>
</body>
</html>
```

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

Список свойств окна, которыми Вы можете управлять:

directories	yes|no	toolbar	yes|no
height	*количество пикселов*	width	*количество пикселов*
location	yes|no	alwaysLowered	yes|no
menubar	yes|no	alwaysRaised	yes|no
resizable	yes|no	dependent	yes|no
scrollbars	yes|no	hotkeys	yes|no
status	yes|no	titlebar	yes|no

Имя окна

Открывая окна, мы должны использовать три аргумента:

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

Второй аргумент — это имя окна. Ранее мы видели, как оно использовалось в параметре target. Так, если Вы знаете имя окна, то можете загрузить туда новую страницу с помощью записи

```
<a href="bla.html" target="displayWindow">
```

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

Обратите внимание, что *myWin* - это вовсе не имя окна. Но только с помощью этой переменной Вы можете получить доступ к окну. И поскольку это обычная переменная, то область ее действия - лишь тот скрипт, в котором она определена. А между тем, имя окна (в данном случае это *displayWindow*) - уникальный идентификатор, которым можно пользоваться с *любого* из окон браузера.

5.2. Закрытие окон

Вы можете также закрывать окна с помощью языка JavaScript. Чтобы сделать это, Вам понадобится метод close().

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

Если теперь в новом окне Вы нажмете кнопку, то оно будет закрыто. open() и close() - это методы объекта window. Однако в нашем случае нет необходимости писать префикс window, если Вы хотите всего лишь вызвать один из методов этого объекта (и такое возможно только для этого объекта).

5.3. Динамическое создание документов

Для начала мы создадим простой HTML-документ, который покажем в новом окне. Рассмотрим следующий скрипт.

```
<html>
<head>
<script language="JavaScript">
function openWin3() {
  myWin= open("", "displayWindow);
  myWin.document.open();
  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>");
  myWin.document.close();
}
</script>
</head>
<body>
<form>
<input type=button value="On-the-fly" onClick="openWin3()">
</form>
</body>
</html>
```

Давайте рассмотрим функцию winOpen3 (). 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, поэтому браузер обязан создать дополнительно новый документ.

В скрипте мы определяем переменную *myWin*. И с ее помощью можем получать доступ к новому окну. Обратите пожалуйста внимание, что в данном случае мы не можем воспользоваться для этой цели именем окна (*displayWindow*).

После того, как мы открыли окно, наступает очередь открыть для записи объект document. Делается это с помощью команды:

```
myWin.document.open();
```

Здесь мы обращаемся к open() - методу объекта document. Однако 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. По завершении этого мы обязаны вновь закрыть документ. Это делается следующей командой:

```
myWin.document.close();
```

Как я уже говорил, Вы можете не только динамически создавать документы, но и по своему выбору размещать их в в том или ином фрейме. Например, если Вы получили два фрейма с именами *frame1* и *frame2*, а теперь во *frame2* хотите сгенерировать новый документ, то для этого в *frame1* Вам достаточно будет написать следующее:

```
parent.frame2.document.open();
parent.frame2.document.write("Here goes your HTML-code");
parent.frame2.document.close();
```

6. Строка состояния и таймеры

6.1. Строка состояния

Составленные Вами программы на JavaScript могут выполнять запись в строку состояния - прямоугольник в нижней части окна Вашего браузера. Все, что Вам необходимо для этого сделать - всего лишь записать нужную строку в *window.status*. В следующем примере создаются две кнопки, которые можно использовать, чтобы записывать некий текст в строку состояния и, соответственно, затем его стирать.

```
<html>
<head>
<script language="JavaScript">
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>
```

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

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

Вы можете спросить, а почему в onMouseOver мы обязаны возвращать результат *true*. На самом деле это означает, что браузер не должен вслед за этим выполнять 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 общем случае, мы всегда можем отменить дальнейшую обработку события браузером, возвращая *true* в своей собственной процедуре обработки события.

6.2. Таймеры

С помощью функции Timeout (или таймера) Вы можете запрограммировать компьютер на выполнение некоторых команд по истечении некоторого времени. В следующем скрипте демострируется кнопка, которая открывает выпадающее окно не сразу, а по истечении 3 секунд. Скрипт выглядит следующим образом:

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

Здесь *setTimeout()* - это метод объекта window. Первый аргумент при вызове - это код JavaScript, который следует выполнить по истечении указанного времени. Во втором аргументе указывается задержка в миллисекундах.

7. Предопределенные объекты

7.1. Объект Date

Давайте рассмотрим объект Date. Судя по названию, он позволяет Вам работать как со временем, так и с датой. Например, Вы можете легко определить, сколько дней еще остается до следующего рождества. Или можете внести в Ваш HTML-документ запись текущего времени.

Так что давайте начнем с примера, который высвечивает на экран текущее время. Сперва мы должны создать новый объект Date. Для этого мы пользуемся оператором *new*:

```
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, это методы - getHours(), setHours(), getMinutes(), setMinutes(), getMonth(), setMonth() и так далее

Чтобы зафиксировать какое-либо другие дату и время, мы можем воспользоваться видоизмененным конструктором

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

Заметьте, что месяцы нумеруются с нуля. Число 1 будет обозначать февраль, ну и так далее.
Теперь мы напишем скрипт, печатающий текущие дату и время.

```
<script language="JavaScript">
now= new Date();
document.write("Time: " + now.getHours() + ":" + now.getMinutes() + "<br>");
if(now.getYear() > 1999)
 document.write("Date:"
   +now.getYear()+"/"+(now.getMonth()+1)+"/"+now.getDate());
else
 document.write("Date:"
   +(1900+now.getYear())+"/"+(now.getMonth()+1)+"/"+now.getDate());
</script>
```

В данном скрипте не выполняется проверки на тот случай, если количество минут окажется меньше, чем 10. Это значит, что Вы можете получить запись времени примерно в следующем виде: *14:3*, что на самом деле должно было бы означать *14:03*. Решение этой проблемы мы рассмотрим в следующем примере.
Рассмотрим теперь скрипт, создающий на экране изображение работающих часов:

```
<html>
<head>
<script Language="JavaScript">
var timeStr, dateStr;
function clock() {
        now= new Date();
        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;
        date= now.getDate();
        month= now.getMonth()+1;
        year= now.getYear();
        dateStr= "" + month;
        dateStr+= ((date < 10) ? "/0" : "/") + date;
        if(now.getYear() > 1999)
          dateStr+= "/" + year;
        else
          dateStr+= "/" + (1900+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>
```

Можно видеть, что функции clock() вызываются программой обработки события onLoad, помещенной в тэг <body>. В разделе body нашей HTML-страницы имеется два элемента формы для ввода текста. Функция clock() записывает в оба эти элемента в корректном формате текущие время и дату.

7.2. Объект Math

Если Вам необходимо в скрипте выполнять математические рассчеты, то некоторые полезные методы для этого Вы найдете в объекте Math. Например, имеется метод синуса sin().

Я бы хотел продемонстрировать работу метода random(). Если Вы вызовете 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):

7.3. Объект Image

С помощью объекта Image Вы можете вносить изменения в графические образы, присутствующие на web-странице. В частности, это позволяет нам создавать мультипликацию.

В JavaScript все изображения предстают в виде массива. Массив этот называется *images* и является свойством объекта document. Например, Вы можете определить, который размер имеет изображение, обратившись к его свойствам *width* и *height*. То есть по записи `document.images[0].width` Вы можете определить ширину первого изображения на web-странице (в пикселах).

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

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

то Вы сможете обращаться к нему, написав `document.myImage` или

```
document.images["myImage"].
```

Начиная с версии JavaScript 1.1, вы имеете возможность назначать новый адрес изображению, уже загруженному в web-страницу. И в результате, изображение будет загружено с этого нового адреса, заменив на web-странице старое. Рассмотрим к примеру запись:

```
<img src="img1.gif" name="myImage">
```

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

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

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

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

```
hiddenImg= new Image();
hiddenImg.src= "img3.gif";
```

В первой строке создается новый объект Image. Во второй строке указывается адрес изображения, которое в дальнейшем будет представлено с помощью объекта *hiddenImg*. Как мы уже видели, запись нового адреса в атрибуте *src* заставляет браузер загружать изображение с указанного адреса. Поэтому, когда выполняется вторая строка нашего примера, начинает загружаться изображение *img2.gif*. Но как подразумевается самим названием *hiddenImg* ("скрытая картинка"), после того, как браузер закончит загрузку, изображение на экране не появится. Оно будет лишь будет сохранено в памяти компьютера (или точнее в кэше) для последующего использования. Чтобы вызвать изображение на экран, мы можем воспользоваться строкой:

```
document.myImage.src= hiddenImg.src;
```

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

Вы можете создать красивые эффекты, используя смену изображений в качестве реакции на определенные события. Например, Вы можете изменять изображения в тот момент, когда курсор мыши попадает на определенную часть страницы:

```
<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>
```