Министерство образования и науки Российской Федерации
КАРАЧАЕВО-ЧЕРКЕССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ У.Д. АЛИЕВА

Практикум по разработке web-приложений на языке JavaScript

УЧЕБНОЕ ПОСОБИЕ

Карачаевск – 2016

УДК-004.43
ББК 32.973.26-018.1

Печатается по решению редакционно-издательского совета Карачаево-Черкесского государственного университета им. У.Д. Алиева

Лепшокова А.Н., Эльканова А.А. Практикум по разработке web-приложений на языке JavaScript / А.Н. Лепшокова, Эльканова А.А. – Карачаевск: КЧГУ, 2016. – 142 с.

Составители: *Лепшокова А.Н.*, к. п. н.,
Эльканова А.А., к. п. н.

ISBN 978-5-8307-0435-9

Изложены основы создания сценариев на языке web-программирования JavaScript. Предназначено для студентов, обучающихся по всем профилям подготовки бакалавров направления: Информатика, Прикладная информатика, Прикладная математика и информатика. Пособие может быть использовано студентами заочной формы обучения.

Рецензенты: *Б.С. Кубекова*, к. ф.-м. н., доцент,
А.М. Узденова, к. ф.-м. н., доцент.

ISBN 978-5-8307-0435-9

Введение

Язык JavaScript отличается от других языков высокого уровня, прежде всего тем, что программы, написанные на нём, не могут выполняться самостоятельно. Они либо встроены в HTML-документ, либо расположены в отдельных файлах и присоединяются к HTML-документу при его загрузке в браузер. Исходные данные берутся из HTML-документа, а результаты выполнения программы возвращаются в него. Программы на JavaScript называют сценариями или скриптами.

Так как сценарии передаются на компьютер пользователя по сети, то от их размера зависит время загрузки сайта. Часто сценарий служит для внесения очень простых изменений в HTML-документ, например, требуется изменить цвет или размер какого-нибудь объекта на экране. Поэтому сценарии, как правило, небольшие, часто состоят из нескольких строк. Формально на их размер никаких ограничений не накладывается.

Наибольшие трудности у разработчика сценариев вызывают необходимость постоянно использовать объектную модель HTML-документа и несовместимость браузеров. Модель состоит из множества объектов, каждый из которых имеет большое количество свойств и методов. Многие свойства имеют большой набор специальных значений. Объекты реагируют на большое число событий. Например, объект div в объектной модели, применяемой в одном из наиболее распространённых браузеров, имеет 51 свойство, 38 методов и 35 событий. Запомнить все необходимые для разработки сценариев сведения об объектной

модели очень трудно, поэтому необходимо использовать специализированный редактор с хорошей справочной системой.

В предлагаемом лабораторном практикуме подробно на примерах рассматриваются основные приёмы программирования на языке JavaScript. Первая часть практикума – справочная. В ней излагаются сведения о языке JavaScript, достаточные для решения задач, предлагаемых в лабораторных работах, описанных во второй части практикума.

Лабораторные работы построены по следующей схеме: описывается проблема, методы её решения, рассматривается пример и предлагается задача для самостоятельного решения.

Большое внимание уделяется проблеме совместимости браузеров. Из-за слишком большой сложности разработки сценариев, предназначенных для работы на браузерах многих типов, рассматривается совместимость только браузеров Internet Explorer 6.0 и Mozilla FireFox 2.0. Все примеры, приводимые в лабораторных работах, проверены на этих браузерах. Часть примеров специально разработана только для одного браузера.

I. Основы создания сценариев

1.1 Способы создания динамических HTML-документов

HTML-документы (страницы), которые видит пользователь Интернета в окне браузера, могут храниться на веб-сервере в виде готовых к использованию файлов, или формироваться по запросу пользователя. Будем называть заранее подготовленные к просмотру страницы статическими. Страницы, формирующиеся «на лету» по запросу пользователя, назовём динамическими.

Типичным является формирование динамической страницы из информации, выбранной из базы данных по запросу пользователя. Например, пользователь просит показать курсы некоторых валют за определённый период времени. Результаты будут выданы в форме таблицы, количество строк и столбцов которой будет зависеть от запроса пользователя. Страница в этом случае формируется на веб-сервере специальной программой, имя которой и параметры для её работы содержатся в адресе (URL), передаваемом браузером. Веб-сервер идентифицирует программу по расширению имени файла и запускает её выполнение. Программа создаёт HTML-документ, который веб-сервер возвращает браузеру в качестве ответа на запрос пользователя.

Желание снизить нагрузку на сеть и на веб-сервер приводит к идее выполнять некоторые преобразования страницы прямо в браузере, например, перемещать рисунок в окне браузера. Для этого в HTML-документ встраивают небольшую программу (сценарий), выполняемую по мере необходимости браузером. Стандартным языком для написания сценариев является JavaScript.

Фирма Microsoft разработала язык сценариев VBScript, не получивший широкого распространения.

Придание динамичности страницам сайтов обеспечивается совокупностью языков HTML и JavaScript. В HTML для расширения возможностей управления видом и расположением элементов на странице стали применяться каскадные таблицы стилей (Cascading Style Sheets или CSS). Браузер при открытии HTML-документа строит его объектную модель (Document Object Model или DOM), используемую как при отображении документа, так и при его изменении скриптом. Организация World Wide Web Consortium (W3C) разработала стандарты на CSS и DOM. Версия HTML 4, поддерживающая стандарты DOM и CSS, называется динамическим HTML (Dynamic HTML или DHTML).

К сожалению, разработчики браузеров не строго придерживаются стандартов, вносят свои элементы в HTML и JavaScript. Скрипт, написанный для конкретной версии какого-либо браузера, может не выполняться даже на других версиях того же браузера, не говоря о браузерах других фирм. Программист, создающий сайт, должен предусмотреть, как будут работать на различных браузерах скрипты, входящие в состав сайта. Кроме того, он должен подумать, как будет выглядеть сайт на браузерах с установленным в целях безопасности запретом на выполнение скриптов.

1.2 Объектная модель документа

Объектная модель документа имеет иерархическую структуру. На вершине этой иерархии находится объект window . Он имеет следующие дочерние объекты:

navigator,

screen,

history,

location,

document.

В скриптах, в основном, используется объект document, так как он целиком описывает загружаемый в браузер HTML-документ. Объекты navigator, screen и history используются для справки о свойствах браузера и компьютера пользователя. Объект location удобно использовать для загрузки страниц или фреймов.

Структура HTML-документа

HTML-документ состоит из элементов (абзацев, гиперссылок, таблиц и т. д.). Элемент обычно делится на три части: начальный тег, содержимое и конечный тег. Название элемента появляется в начальном теге (<название_элемента>) и в конечном теге (</название_элемента >). Пример элемента:

<H1> Структура HTML-документа</H1>

Начальный тег может содержать атрибуты, определяющие свойства элемента. Для некоторых элементов допускается отсутствие конечного тега (например, для тегов P, TR, TD). Некоторые типы элементов HTML-документа не имеют

содержимого. Например, обрыв строки BR не имеет содержимого, его единственная задача - обозначить конец строки текста. Такие пустые элементы никогда не имеют конечного тега.

Элементы HTML-документа могут входить друг в друга, образуя сложную иерархическую структуру.

1.3 Структура объекта document

Объект document ставится в соответствие всему HTML-документу и создаётся браузером при его загрузке. Всем элементам HTML-документа ставятся в соответствие объекты, входящие в состав объекта document, т. е. структура объекта document повторяет структуру порождающего его HTML-документа. Атрибутам элемента соответствуют свойства объекта, имеющие те же названия, что и атрибуты.

В HTML-документе может быть несколько элементов с одинаковым названием, например, контейнеров DIV . Поэтому каждый элемент имеет атрибут ID – идентификатор элемента. Объекту, соответствующему элементу, автоматически присваивается ID этого элемента. Кроме того, объекты в модели нумеруются по порядку их появления в модели. Использовать номера объектов нужно осторожно, так как при внесении иэменений в документ количество и порядок объектов могут меняться.

Для идентификации некоторых элементов (FORM, IMG) кроме атрибута ID используется атрибут NAME. Области действия ID и NAME совпадают лишь частично. Рекомендуется в случае

одновременного применения обоих атрибутов присваивать им одинаковые значения.

1.4 События

Изменения в динамических страницах могут происходить либо циклически, либо в ответ на действия пользователя. Например, чтобы обратить внимание на важную информацию, можно с помощью скрипта периодически менять цвет соответствующей области экрана. Примером реакции на действия пользователя служит увеличение размера рисунка после щелчка по нему мышкой. Во всех случаях сигналом к изменениям служит какое-то событие. В динамическом HTML события включены в число атрибутов элементов. Следующие события используются большинством элементов:

onclick возникает, когда указательное устройство щёлкает на элементе;

ondblclick возникает, когда указательное устройство дважды щёлкает на элементе;

onmousedown возникает, когда кнопка указательного устройства "нажала" на элемент;

onmouseup возникает, когда кнопка указательного устройства отпущена над элементом;

onmouseover возникает, когда указательное устройство проходит над элементом;

onmousemove возникает, когда указательное устройство перемещается в тот момент, когда находится над элементом;

onmouseout возникает, когда указательное устройство убирается с элемента;

onkeypress возникает, когда клавиша нажата и отпущена над элементом;

onkeydown возникает, когда клавиша нажата над элементом.

onkeyup возникает, когда клавиша отпущена над элементом.

Указательным устройством, как правило, служит мышка.

Рассмотрим пример простейшей динамической страницы. Допустим, что в браузере после загрузки страницы видно только одно слово Письмо. При щелчке по этому слову всплывает маленькое окошко с сообщением Это заголовок. HTML-код этой страницы:

<html> <H1 onclick= "window.alert('Это заголовок')"> Письмо</h1> </html>

При загрузке этого документа браузер добавит в объектную модель объекты HEAD и BODY. Вставленный в заголовок H1 атрибут onclick вызовет при щелчке по слову Письмо метод alert, в результате выполнения которого появится окно с сообщением. Так как объект window находится в вершине объектной модели документа, то его можно не указывать. В рассматриваемом примере весь скрипт поместился внутри тега. Для размещения сложных скриптов служат специальные средства языков DHTML и JavaScript.

1.5 Переменные

Имя (идентификатор) переменной может состоять из букв, цифр и знаков $ и _ (подчёркивание). Имя переменной не должно

начинаться с цифры. В имени различаются строчные (маленькие) и прописные (большие) буквы. Например, Alfa, alfa и aLFa – разные переменные. В качестве имён переменных нельзя использовать ключевые слова языка JavaScript. Переменные могут иметь следующие типы:

целые числа (integer);

числа с плавающей запятой (float);

строки (string);

логические (booleans);

объекты.

В JavaScript тип переменной явно не объявляется. Создают переменную чаще всего, присваивая ей начальное значение:

x = 2; // или var a = 2

В приведённом примере две косые черты (//) указывают на комментарий. Начало и конец многострочного комментария обозначаются символами /* и */. После оператора можно ставить точку с запятой. Если в одной строке записано несколько операторов, то точка с запятой отделяет операторы друг от друга. Можно создать переменную без присвоения ей начального значения, например,

Var y;

Ключевое слово var используется для определения области видимости переменных. Переменная, созданная в функции с использованием var, видна только в этой функции и называется локальной. Во всех остальных случаях переменная является глобальной и видна во всей программе, включая функции. Пример:

<HTML>Скрипт

```
<SCRIPT> var x = 2 //глобальная переменная
y = 3*x //глобальная переменная
z= kvadr(y)
alert( "z="+z +" x ="+x)// z=36 x=2
function kvadr(a)
{ var x = a*a //x - локальная переменная, действующая
//только в kvadr(a)
 Return x //x=36 }
</script>
</html>
```

Тип переменной определяется (почти так же как в языках Perl и PHP) из контекста программы. В приведённом примере использованы операторы присваивания в стиле языка C (подробнее в разделе Операторы).

1. a = 2; // целое
2. a + = 1.5 // 2 + 1.5 = 3.5 - число с плавающей запятой
3. a * ="4" //3.5*4=14
4. a + = "2" //конкатенация строковых величин "14" и "2".
//Результат "142"
5. b = 53; //целое число
6. c = " года. ";
7. d = b + c //строка "53 года. "

В строках 4 и 7 знак + использован для обозначения операции конкатенации (слияния) строковых переменных, а не сложения двух чисел. Одинаковое обозначение двух разных операций

приводит к необходимости использовать функции явного преобразования переменной из строкового типа в тип числа с плавающей запятой (функция parseFloat()) или в тип целого числа (функция parseInt()). Пример использования функций parseFloat() и parseInt():

1. a ="3.7";
2. b = "6.1";c = a * b // результат 3.7 * 6.1 = 22.57
3. c=a+b; // результат "3.76.1"
4. d=parseFloat(a)+parseFloat(b); // результат 3.7 + 6.1=9.8
5. e=parseInt(a)+parseInt(b); // результат 3 + 6 = 9

В строке 3 переменные a и b перемножаются и поэтому автоматически преобразуются в числа с плавающей запятой. В строке 4 выполняется операции конкатенации, так как в строках 1 и 2 переменным a и b были присвоены строковые значения. В строках 5 и 6 явно задаётся преобразование переменных *a* и *b* в числовой тип, поэтому выполняется операция сложения.

1.6 Операторы

В JavaScript, как и в большинстве языков программирования, используются традиционные математические операторы и операторы, применяемые только в программировании. Математические операторы делятся на три группы: арифметические, операторы сравнения и логические. Среди арифметических операторов (табл. 1) в JavaScript отсутствует возведение в степень.

Таблица 1. Арифметические операторы

Название	Обозначение	Пример
Сложение	+	a + b
Вычитание	-	a - b
Умножение	*	a * b
Деление	./	a / b
Деление по модулю	%	a % b
Увеличение на 1 (инкремент)	++	a++
Уменьшение на 1 (декремент)	--	a--

Возведение числа A в степень N выполняется так:

x = Math.pow(A,N), где pow(A,N) – метод объекта Math.

Операторы сравнения (табл. 2) применяются к числовым, строковым и логическим выражениям.

Таблица 2. Операторы сравнения

Название	Обозначение	Пример
Равно	==	a == b
Не равно	!=	a != b
Больше	>	a > b
Меньше	<	a < b
Больше или равно	>=	a >= b

Меньше или равно	<=	a <= b

В JavaScript используются три логических оператора: И, ИЛИ и НЕ, имеющие обозначения &&, || и ! соответственно. Результат логической операции может иметь одно из двух значений: true (истина) и false (ложь). Логической величине true соответствует число 1, a false– 0. Примеры логических операций:

a = 2; b = 3; st = "a"; // исх. данные для лог.выражений

a > b || st == "a" // истина

a > b || st > "a" // ложь

a > b && st == "a" // ложь

a < b &&

st == "a" // истина

!(a < b) // ложь

Операторы, применяемые только в программировании, также можно разделить на три группы: присваивания, условного перехода и циклические.

Примеры операторов присваивания:

a = 5 //записать число 5 в область памяти,

//отведённую под переменную a

a + = 2 //вместо оператора a = a + 5

a - =2;

a * = 2;

a / = 2;

a % = 2

Результаты действия инкремента и декремента зависят от того, как они расположены относительно операнда:

a = b++; // сначала a присваивается значение b, затем b увеличивается на 1

a = ++b; //сначала b увеличивается на 1, затем a присваивается значение b

Примеры операторов условного перехода if и switch:

```
Var a=3
If (a > 2) alert("a больше двух");
a = 1;
b = -3;
c = 2;
det = Math.pow(b,2) - 4*a*c
if (det < 0)
{ alert("действительных корней нет")
  r = 0 }
else if (det == 0)
{ alert("корни одинаковые")
  r=1 }
else
{ alert("два корня")
  r=2}
switch (a)
{ case 0:
    alert("ноль")
    break
case 1:
    alert(1)
```

```
    break
case 2:
    alert("два")
    break
default:
    alert("не подходит"); }
```

Если в операторе switch убрать ключевое слово break, то будут выполнены все операторы после первого совпадения.

Вместо условного оператора if может использоваться следующая конструкция:

переменная = (условие)? значение 1 : значение 2

Например, в результате выполнения оператора

x = (4 > 2) ? "правильно" : "ошибка"

переменная x примет значение правильно.

Примеры применения операторов цикла for, while и do while

```
// Оператор for. Сложение всех нечётных чисел от 1 до 10
S = 0;
for(i= 1; i <= 10; i++) {S += i}
S = 0 //сложение чисел 5 ,10,...,25
for(i = 5; i<= 25; i+=5) {S += i}
// Оператор while. Сложение всех нечётных чисел от 1 до10
S = 0; i = 0;
while(i < 10) {S += ++i}
// Оператор do ... while.
//Сложение всех нечётных чисел от 1 до 10
S = 0; i = 0;
```

```
do
{ S += ++i}
while(i < 10)
```

1.7. Массивы

Массивы в JavaScript одновременно рассматриваются и как объекты и как массивы в традиционном понимании. Способы создания массивов показаны в следующих примерах:

```
var a = new Array(5) //создан массив из пяти элементов
var b = new Array(2,7,12) // создан массив b из трёх
//элементов и его элементам присвоены значения
```

Нумерация элементов массива начинается с нуля. Для обращения к элементу массива используются квадратные скобки, например

```
x = b[1] // x равен 7
```

Длина массива определяется как его свойство:

```
c = a.length // длина массива a c= 5
```

Элементы одного массива могут иметь разный тип. В следующем примере массив состоит из строковых и числовых элементов.

```
earth = new Array(4)
earth[0]= "Планета" // строка
earth[1]= "24 часа" // строка
earth[2]= 6378 // целое число
earth[3]= 365.25 // число с плавающей запятой
```

Информацию о планете Земля удобно хранить в ассоциативном массиве.

```
earthA = new Array(4)
earthA.type_= "Планета"
earthA.day_= "24 часа"
earthA.radius= 6378
earthA.period = 365.25
```

Слова type и day – ключевые в JavaScript. Поэтому к идентификаторам переменных type_ и day_ добавлен знак подчёркивания. Для извлечения из обоих массивов радиуса Земли служат операторы:

```
R=earth[2]
R= earthA.radius
```

Элементами массива могут служить массивы:

```
product = new Array()
product[0] = new Array("Пшеница",250)
product[1] = new Array("Рожь",300)
product[2] = new Array("Ячмень",200)
```

Массив product – двумерный. Элемент product[0][1] имеет значение 250. Длина двумерного массива определяется свойством length:

```
L = product.length // L = 3
m = product[0].length // m = 2
```

Элементами массива product являются три массива, каждый из которых состоит из двух элементов.

1.8 Строки

Строка в JavaScript является одновременно и объектом string и переменной, поэтому может быть создана двумя способами:

st1 =new String("Строка – это объект")
st2 = "Строка – это переменная"

Строки, рассматривая их как переменные, можно объединять с помощью операции конкатенации. Рассмотрим пример, в котором новая строка получается в результате объединения двух строковых переменных, строковой константы и числовой переменной:

st1 = "Санкт-Петербург основан" n = 1703

st2 =" году"

st3= st1 + " в " + n +st2

В строке st3 будет предложение Санкт-Петербург основан в 1703 году. Тип переменной n изменён автоматически. Наиболее часто над строкой совершаются следующие действия:

- ищется подстрока;
- выделяется несколько символов (подстрока), начиная с символа с заданным номером;
- заменяется одна подстрока другой.

Для поиска подстроки используется метод indexOf(), имеющий следующий синтаксис:

строка.indexOf(подстрока [,начало]) ,

//где начало – номер символа в строке, с которого нужно

начинать поиск.

Результат применения метода indexOf() – номер символа, с которого начинается искомая подстрока. Нумерация символов начинается с нуля. Найдём в строке st3 номер символа, с которого начинается подстрока Петербург:

n = st3.indexOf("Петербург") // результат – 6

Для выделения нескольких символов используется метод substr(), имеющий следующий синтаксис:

строка.substr(начало [,длина]) ,

где: начало – номер символа, с которого начинается

подстрока, длина – длина искомой подстроки

Результат применения метода – найденная подстрока. Выделим в st3 слово Петербург: gorod = st3.substr(6,9)

Для замены подстроки используется метод replace(), имеющий следующий синтаксис:

строка.replace(регулярное выражение, заменяющая подстрока)

В качестве регулярного выражения в простейшем случае можно применить заменяемую подстроку. Заменим в строке st3 подстроку 1703 году на подстроку XVIII веке:

osnovanie=st3.replace(/1703 году/," XVIII веке")

Строка st3 примет значение Санкт-Петербург основан в XVIII веке.

1.9 Объекты Math и Number

Свойствами объекта Math служат математические константы, а методами - математические функции. Примеры использования объекта Math:

r = 5 //радиус. Как объект:

r = new Number(5)

L = 2 * Math.PI * r //Длины окружности L=31.41592653589793

Lc =Math.ceil(L) // Lc=32. Округление до ближайшего

// большего или равного целого

Ld = Math.floor(L) // Ld=31.Округление до ближайшего

// меньшего или равного целого

Lok =Math.round(L)//Lok=31. Округление до ближайшего целого

S= Math.PI * Math.pow(r,2) //Площадь круга

Все переменные в примере являются объектами Number. Общий вид выражения для создания объекта Number:

переменная = new Number(число)

Создание конкретного объекта:

z = new Number(43.567)

Часто используемые свойства объекта Number:

L = new Number(31.41592653589793)

//L.toFixed(n) - n знаков после запятой

L_3 = L.toFixed(3) // L_3=31.416

//L.toPrecision(n) - n значащих цифр

L4 = L.toPrecision(4) //L4=31.42

Глава II. Обзор возможностей языка (Концепция и разработка сценариев)

2.1. Вывод текста на Web-страницу

JavaScript представляют собой OOP (Object Orientated Programming, объектно-ориентированный язык программирования). JavaScript не может существовать сам по себе, он должен выполняться внутри Web-страницы, а Web-страница должна просматриваться в браузере, который понимает язык JavaScript . JavaScript располагается внутри документа HTML и сохраняется в виде текста вместе с документом HTML. У него четкая форма. И пренебрегать ею нельзя.

Концепция

Рассмотрим сценарий, который выводит текст на Web-страницу:

```
<SCRIPT LANGUAGE="javascript">
document.write
("<FONT COLOR='RED'>Это красный текст </FONT>")
</SCRIPT>
```

Результат работы сценария

Это текст красного цвета.

Разбор сценария

<SCRIPT LANGUAGE="JavaScript">

Это код HTML, который дает браузеру понять, что с этого места начинается JavaScript. Любой сценарий JavaScript начинается с такой команды. Команда LANGUAGE (язык) = "JavaScript" не даст браузеру запутаться, так как есть еще и другие типы сценариев, например, VBS или LiveScript. Код </SCRIPT>

заканчивает любой сценарий JavaScript без исключений. Это основная часть сценария:

document.write("<FONT COLOR='RED'>Это красный текст </FONT>")

Сценарий устроен следующим образом. С помощью DOCUMENT объявляется документ (документ HTML). Этот документ будет изменен — в нем что-то будет написано (WRITE). То, что будет написано, находится внутри скобок. DOCUMENT представляет собой объект. Слово WRITE (писать), отделенное точкой, называется методом объекта. Таким образом, сценарий по сути говорит: "Возьмите объект (что-то, уже существующее) и что-то в нем запишите". Текст внутри скобок находится в кавычках. В HTML эти кавычки не требуются. Здесь они необходимы. Если внутри кавычек встречаются еще кавычки, то они должны быть разного типа. Текст в кавычках представляет собой простой код HTML. В нем команда <FONT> делает текст красным. RED находится в одинарных кавычках. Если использовать двойные, JavaScript решит, что это конец строки, и получится, что только часть текста будет записана в объект, и это будет ошибкой.

Таким образом, HTML перекрасил текст в красный цвет, а JavaScript записал код на страницу.

Задание

Измените сценарий так, чтобы выводились две строки текста, красная и синяя. Но это надо сделать с помощью дополнительных

команд JavaScript, а не просто добавить код HTML к приведенному примеру. На странице должно выводиться следующее:

Это красный текст

Это синий текст

2.2. Дата и время

Нам известно, что существует такой объект, как документ. Иначе в нем ничего нельзя было бы написать. Рассмотрим семь новых методов: getDay(), getDate(), getMonth(), getYear(), getHours(), getMinutes(), и getSeconds() (получить День, Число, Месяц, Год, Час, Минуту, Секунду). Все они уже существуют, их можно взять и поместить на Web-страницу. Проблема в том, что это всего лишь методы. Для действия им нужен объект, а документ для этих целей не годится, необходимо создать объект.

Сценарий

```
<SCRIPT LANGUAGE="JavaScript">
//Сценарий выводит точную дату и время посещения страницы
Now = new Date();
document.write("Сегодня " + Now.getDate()+
"-" + (Now.getMonth()+1) + "-" + Now.getFullYear()
+ ". Вы зашли на Web-страницу ровно в: " + Now.getHours() +
":" + Now.getMinutes() + " и " + Now.getSeconds() +
" секунд.")
</SCRIPT>
```

Строка document.write не должна прерываться. Она разбита на несколько строк, чтобы удобнее было читать.

Результат работы сценария

Сегодня 18-7-2006. Вы зашли на эту Web-страницу ровно в: 8:42 и 3 секунд.

Разбор сценария

Строка document.write уходит далеко за границы экрана. Эту форму необходимо сохранить. Если разбить эту строку на две, браузер выдаст сообщение об ошибке. Двойная косая черта указывает на комментарий внутри сценария. Она означает, что следующий за ней текст не будет использоваться в процессе. Количество строк комментария ничем не ограничено, надо только помнить, что каждая строка должна начинаться с двойной косой черты //.

Методы Date (Дата) и Time (Время)

Если посмотреть сценарий, то можно видеть, что результат создается командой записи в документ числа, месяца, года, часа, минуты и секунды.

Каждый из этих объектов был создан с помощью метода в виде getЧто-либо(). Обратите внимание на заглавную букву. Сначала "get" в нижнем регистре, потом слово с Заглавной буквы, описывающее объект.

Следует запомнить, что каждый из этих объектов является числом. Все методы возвращают только числа. Даже метод getDay(), который возвращает день недели, выражается числом от единицы до семи.

Начнем с месяца. getMonth() — это метод, отвечающий за месяц. Теперь необходимо определиться, методом какого объекта является getMonth(). Метод действует на объект.

Может показаться, что get Что-либо() — это метод объекта document. Метод документа — write. getMonth() на самом деле является методом объекта Date. Объект Date задается в команде:

Now = new Date();

С этим объектом будет работать метод getMonth(). Имея дело с датой и временем, всякий раз пользуйтесь той же схемой. Прежде всего необходимо создать объект. В данном случае объект называется Now , может называться Zork или Fred, браузеру все равно. Это не имеет значения, если объект получает оригинальное имя, которое больше нигде в JavaScript не встречается.

Команда говорит: Now — это объект, который представляет new Date() (новую Дату). Дата обязательно должна быть новой. Таким способом вы будете получать новую дату каждый раз, когда заходите на страницу или обновляете ее. Без команды new дата будет оставаться статичной.

Точка с запятой в конце строки действует как признак конца оператора. Она указывает на то, что строка JavaScript закончена. Без нее браузер решил бы, что команда продолжается на следующей строке, а это ошибка.

Итак, у нас есть объект, на который может воздействовать метод getMonth(). Нам нужно напечатать месяц на странице,

значит, где-то должна быть команда document.write(). Мы знаем также, что текст в скобках будет выведен на страницу, поэтому давайте напишем все это, соблюдаю последовательноть.

Сначала пишем <SCRIPT LANGUAGE="javascript">.

Затем вставляем комментарий о том, для чего предназначен сценарий.

Прежде чем можно будет обратиться к getMonth(), необходимо создать объект, затем поместить оператор document.write. Текст в скобках после document.write оформляем по правилам.

Текст, выводимый на странице, должен быть окружен двойными кавычками (одинарные кавычки для кода HTML внутри двойных). Сочетание текста и команд требует знака "плюс" + между элементами.

Объект и метод разделены точкой, так что команда "поместить месяц" должна выглядеть так: Now.getMonth().

Следует иметь ввиду, что Now.getMonth() — это не текст, который должен быть виден на странице, а команда, которая указывает месяц. Поэтому не нужно ставить ее ни в какие кавычки. Обратите внимание, что к getMonth добавлена 1: (Now.getMonth()+1). Причина в том, что возвращаемые месяцы ведут отсчет от 0, поэтому для правильного вывода месяца необходимо добавлять 1. Прибавление единицы помещено в скобки (и), чтобы знак + не создавал путаницы со знаками +, соединяющими выводимые элементы. Заканчиваем командой </SCRIPT>.

Вот что получилось:

```
<SCRIPT LANGUAGE="javascript">
//Сценарий выведет на странице номер месяца
Now = new Date();
document.write("Сейчас месяц " + (Now.getMonth()+1))
</SCRIPT>
```

Следует помнить, что дефис должен быть виден на странице, поэтому его следует ставить в кавычки. Все части связаны между собой знаком плюса +.

При этом сколько бы пробелов вы не вставили до и после знака плюс, это никак не повлияет на видимый результат. Элементы пойдут сплошным текстом. Поэтому, если в текст требуется вставить пробелы, следует добавлять их в части текста в кавычках. Например:

```
"Сейчас ровно   "
```

Здесь добавлены два пробела перед второй кавычкой. Это превратится в два пробела на странице, когда сценарий выполнится. Разберем строку с датой. Это выглядит следующим образом:

```
document.write("Сегодня "
+  (Now.getMonth()+1)+
"-" + Now.getDate() + "-"
+ Now.getFullYear() + ".
Вы зашли на Web-страницу ровно в : "
+ Now.getHours() +
":" + Now.getMinutes()  + " и "
```

```
+ Now.getSeconds() +
" секунд")
```

Начинаем с "Сегодня ", прибавив в конце пробел. Затем следует знак плюс. (Now.getMonth() + 1) добавляется без кавычек, потому что нам нужен не этот текст, а возвращаемое число. Еще плюс. Потом дефис в кавычках, чтобы отделить следующее число. Никаких пробелов, потому что они должны стоять вплотную. Плюс. Потом Now.getDate() без кавычек, чтобы у нас был день. Плюс. Еще дефис в кавычках, чтобы он был виден на странице. Плюс. Еще один метод Now.getFullYear сообщит год. Продолжаем дальше по этой схеме, и сценарий выведет время.

Если использовать просто getMonth(), то номер месяца будет на единицу меньше чем нужно, так как числа JavaScript считает от нуля. То есть, январь нулевой месяц и так далее.

Для того чтобы избежать этой ошибки надо прибавить 1. Нужно ввести несколько переменных, то есть присвоить имя некоему элементу. Присваиваете new Date() имя, как уже делали раньше. Затем присваиваете имя егомро (Месяц Плюс Один) коду, который вызывает месяц. И прибавляете к этому имени единицу. mpo1 – новая команда Это выглядит следующим образом:

```
<SCRIPT LANGUAGE="javascript">
RightNow = new Date();
var mpo = RightNow.getMonth();
var mpo1 = mpo + 1;
document.write("Сегодня месяц " +mpo1+ ".");
</SCRIPT>
```

Получилось:

Сегодня месяц 8.

А это уж правильный месяц.

Задание

Напишите сценарий, который выводит на Web-странице дату, разделенную косой чертой. Приветственный текст должен быть зеленого цвета. Добавьте также комментарий о том, что это вы написали этот сценарий.

2.3. Обработчики событий: onMouseOver

Мы обсудили объекты и методы. Теперь приступим к рассмотрению событий (events). События (event) и обработчики событий (event handler) относятся к JavaScript, но они скорее "встроены" в HTML-код, а не существуют самостоятельно, как те сценарии, которые мы создали. События являются встроенными, так что они не требуют команд <SCRIPT> и </SCRIPT>. Сами они являются не сценариями, а скорее небольшими интерфейсами, обеспечивающими взаимодействие между страницей и читателем.

События — это то, что происходит. Они добавят динамики Web-сайту. Существует множество событий, с которыми мы познакомимся, но для начала выберем одно из наиболее популярных — onMouseOver (навести курсор мыши).

Сценарий

```
<A HREF="http://www.index.ru"
onMouseOver="window.status='Почтовая служба';
return true">Ссылка</A>
```

<A HREF="http://www.adress.ru" onMouseOver ="document.bgColor ='pink'; window.status='Посетите ADRESS.RU'; return true;">Щелкните здесь</A>

При наведении курсора мыши на ссылку строка состояния в окне браузера изменится. Разберем сценарий и попробуем его как-нибудь изменить.

Во-первых, onMouseOver представляет собой обработчик событий (Event Handler) гипертекстовой ссылки. Он используется внутри гиперссылки.

Формат гипертекстовой ссылки остается без изменений. Те же команды и те же двойные кавычки. Обработчик событий onMouseOver ставится сразу же после адреса URL.

Событие (Event) приводится в действие, когда браузер распознает onMouseOver="". Общая схема уже должна быть немного понятна: два элемента, разделенные точкой. До сих пор это означало, что один является объектом, а другой методом. Но в данном случае объектом является window (окно), оно существует;status (статус) представляет собой свойство (property) окна. Это небольшой участок окна, где должен разместиться следующий далее в команде текст. Это проще запомнить, если представить, что метод обычно выражается глаголом, как write (писать) или get (получить). Свойство выражается существительным и существует как небольшая часть элемента, стоящего перед точкой.

После window.status ставится знак равенства =, за которым следует то, что должно произойти. В данном случае это текст в

одинарных кавычках. Он появится в строке состояния, когда вы наведете курсор на гипертекстовую ссылку.

Затем следует return true. Эти два слова имеют вполне определенное влияние на то, что произойдет, когда указатель мыши переместится на ссылку. Если они присутствуют, сценарий проверит, есть ли строка состояния. Если проверка будет успешной (true), происходит событие. Когда указатель мыши перемещается на ссылку, то текст в строке состояния блокируется. Он больше не изменяется при последующих перемещениях указателя мыши на ссылку. (Если обновить страницу, то можно будет это увидеть).

Если проверка строки состояния отсутствует, то выполняется действие по умолчанию. По умолчанию в HTML выводится URL-ссылка, на которую указывает курсор мыши. Затем, когда курсор смещается со ссылки, произойдет событие. И так как проверка отсутствует, то событие будет происходить всякий раз при перемещении курсора мыши над ссылкой.

Если свойства есть у окна, то другие объекты тоже должны иметь свойства. Например, цвет фона. В HTML цветом фона страницы управляет команда BGCOLOR. В JavaScript цвет фона обозначается bgColor. Создадим ссылку, которая изменяла бы цвет фона окна с помощью события onMouseOver. Раз это ссылка, то сохраним тот же формат, который использовался ранее.

Значит, это и есть нужный объект. Заменим window на document. Мы собираемся изменить фоновый цвет объекта, потому заменим status на bgColor. Появление текста нам больше не

нужно, поэтому заменим его цветом. Возьмем розовый (pink). Нам нужно, чтобы новый цвет оставался независимо от того, сколько раз мы будем наводить курсор на ссылку, потому вставляем return true после точки с запятой.

```
<A HREF="http://www.ADRESS.ru"
onMouseOver="document.bgColor='pink';
return true">Щелкните здесь</A>
```

Нужно написать две команды onMouseOver. Попробуем это реализовать. Эти две команды не разделяются. Мы хотим, чтобы они произошли одновременно, поэтому не будем разделять команды точкой с запятой, так как точка с запятой означает конец оператора.

Новое правило: ставьте запятую, чтобы отделить друг от друга несколько событий JavaScript.

В кавычки помещают отдельные элементы вроде текста. Раз нам нужно, чтобы обе команды выполнились одновременно, как одна, то ставим кавычки в самом начале первой и в самом конце второй. Таким образом, мы показываем браузеру, что все это одно событие.

Вот что мы получим:

```
<A HREF="http://www. ADRESS.ru"
onMouseOver="document.bgColor='pink',
onMouseOver=window.status='Посетите ADRESS.RU';
return true">Щелкните здесь</A>
```

Таких обработчиков событий великое множество, и все они прекрасно работают.

Задание

В этом задании предлагается воспользоваться новым методом, alert() (предупредить). Он выводит небольшое диалоговое окно с текстом и кнопкой OK. Попробуйте сделать так, чтобы окно предупреждения появлялось при наведении курсора на ссылку. Вот формат команды:

alert('выводимый в окне текст')

Продумайте решение, решите, что должно произойти сначала, что потом.

2.4. Обработчики событий: onMouseOver

Мы знаем, что команда onMouseOver запускает событие, если навести курсор на ссылку. Аналогично, щелкнув по ссылке, можно с таким же успехом запустить событие с помощью команды onClick.

Чтобы продемонстрировать действие команды, воспользуемся методом alert. Вот его схема: alert('текст, который появится в окне')

Таким образом, применяя тот же подход, что и для onMouseOver, получаем код:

<A HREF="http://www.mail.ru"

onClick="alert('Посмотреть почту!');">

Щелкните здесь</A>

При щелчке на ссылке появится окно с сообщением "Посмотреть почту!"

Помните, что внутри одинарных кавычек нельзя употреблять слова с апострофами ', иначе браузер поймет их как окончание текста, а это будет ошибка.

Команда onFocus

Это обработчик событий, который вызывает действие, когда пользователь "фокусируется" на элементе страницы. Он будет работать для элементов формы: флажков, текстовых полей, текстовых областей и др.

Вот пример:

```
<FORM>
<INPUT TYPE="text" SIZE="30"
onFocus="window.status='Текст в строке состояния';">
</FORM>
```

При использовании этого сценария выводится текстовое поле, при щелчке в котором в строке состояния выводится строка 'Текст в строке состояния'.

Команда onBlur

Если можно направить фокус на объект, значит, можно и "потерять фокус". Обработчик событий onBlur позволяет сообщить пользователю о том, что он изменил свой ответ. Приведем пример. Создается текстовое поле с текстом. Надо изменить текст и затем щелкнуть мышью вне поля, имитируя переход фокуса к другому элементу:

```
<FORM>
<INPUT TYPE="text" SIZE="40"
VALUE="Впишите свое имя и щелкните вне текстового поля"
onBlur="alert('Вы изменили ответ — вы уверены, что он правильный?');">
</FORM>
```

Команда onChange

Действие этой команды очень похоже на действие предыдущей, onBlur. Ее главная задача — проверка. Этот обработчик события проверяет, сделал ли пользователь то, о чем его просили. Пример очень похож на предыдущий, но действует по-другому.

```
<FORM>
<INPUT TYPE="text" SIZE="45"
VALUE="Измените текст и щелкните вне поля — затем
 проверьте строку состояния"
onChange="window.status='Текст был изменен'">
</FORM>
```

Т.е. при изменении текста в поле и последующем изменении фокуса в строке состояния выводится сообщение 'Текст был изменен'.

Событие onSelect

Эта команда работает так же, как и три предыдущие, указывая, что в поле ввода формы произошли изменения. Отличие состоит в том, что данная команда реагирует, когда в поле ввода что-то было выделено.

Команда onSubmit

Это очень популярная команда. Она позволяет вызвать какое-либо действие, когда нажимаете кнопку Submit (отослать, отправить). Когда пользователь нажимает на эту кнопку, команда выведет на экран страницы текст: "Спасибо, что вы нам написали". Формат команды следующий:

<FORM>

<INPUT TYPE="submit"

onSubmit="parent.location='спасибо.html'";>

</FORM>

Здесь используются новые команды. parent.location — это стандартная схема ссылки на другую страницу. В данном случае parent является свойством окна браузера, а location — объектом, который должен появиться в этом окне. То есть parent.location=" означает ссылку.

Команды onLoad и onUnload

Обе они помещаются внутрь команды <BODY> документа HTML. Они вызывают событие, когда страница открывается или закрывается, то есть когда пользователь уходит со страницы. Эти команды будут очень полезны при работе с функциями.

Задание

В этом задании предлагается создать форму, которая будет взаимодействовать с пользователем (для этого необходимо знать о командах формы).

Форма должна иметь три элемента: поле ввода с просьбой ввести имя; два поля для флажков с вопросом о том, что предпочитает пользователь — мороженое или шоколад; кнопку отправки данных (submit).

С каждым элементом должно произойти следующее.

При вводе имени в строке состояния должны появиться слова: Введите свое имя.

Два поля с флажками должны отослать в строку состояния слова: Вы выбрали... и выбор пользователя.

При нажатии на кнопку отправки должно появиться окно с благодарностью пользователю за участие в опросе.

2.5. Запрос пользователю и переменные

Концепция

В этом уроке рассматриваются две концепции. Одна из них используется, когда необходимо запросить у пользователя информацию. Вторая — создание переменных — будет постоянно применяться при работе с JavaScript.

Мы просим пользователя ввести имя, и с этим именем будет связана переменная. Когда переменная будет присвоена, мы сможем ввести ее в строку document.write, которая выведет имя пользователя на странице.

Сценарий

```
<SCRIPT type="text/javascript">
/* Этот сценарий предназначен для получения информации
от пользователя и вывода ее на странице */
var user_name = prompt ("Введите свое имя в поле ниже","Здесь");
document.write("Привет, " + user_name + "!
Добро пожаловать на мою страницу!");
</SCRIPT>
```

Текст в скобках должен располагаться на одной строке.

Результат работы сценария

Привет, OL! Добро пожаловать на мою страницу!

Разбор сценария

Создание переменной. Переменные имеют первостепенное значение в JavaScript. Необходимо знать, как их создавать. Для вывода функции JavaScript задается имя, состоящее из одного слова. Вот строка из скрипта, которая задает переменную:

var user_name = prompt ("Введите свое имя в поле ниже","Здесь")

Переменная была создана по следующей схеме.

var (от variable, переменная) объявляет, что следующим словом будет имя переменной.

user_name (имя_пользователя) — имя переменной. Произвольное. Оно не обязательно должно быть таким длинным. Можно было бы использовать при желании просто N. Но удобнее называть переменные так, чтобы легко было вспомнить, о чем идет речь.

Следует помнить, что регистр имеет значение для JavaScript, следовательно, если переменная обозначена Dog, то буква D каждый раз должна быть заглавной, иначе браузер посчитает их за два разных слова.

Здесь нет никаких кавычек, просто надо ставить одно слово за другим, как показано выше.

Знак равенства = указывает на то, что переменная будет равна результату следующей команды.

В данном случае переменная будет представлять результат, полученный с помощью окна запроса

Команда Prompt

В данном примере используется новая команда prompt (запрос). Этот метод выводит окно с сообщением и полем ввода. Вот формат запроса:

var variable_name = prompt("Текст окна","Текст в поле ввода")

Можно видеть, что var и присваиваемое имя переменной включены в формат. Иначе получился бы запрос, но ничего нельзя было. Теперь, зная все составляющие блоки, вернемся снова к сценарию:

var user_name = prompt ("Введите свое имя в поле ниже","Здесь");

document.write("Привет, " + user_name +

"!Добро пожаловать на мою страницу!");

Ниже представлен весь процесс.

Имя переменной user_name присвоено результату запроса.

prompt просит пользователя написать свое имя в поле ввода.

В поле ввода записано: "Здесь."

Точка с запятой в конце строки.

document.write вызывает текст "Привет, ".

Знак плюс + указывает, что все элементы идут друг за другом.

user_name содержит результат запроса. Еще плюс.

"!Добро пожаловать на мою страницу!" завершает текст.

Точка с запятой.

Задание

Необходимо выполнить анализ показанного ниже сценария, объяснив, как работает каждая его часть.

Задание Самостоятельный разбор сценария

Вот сценарий. Разберите его на составляющие элементы.

```
<SCRIPT type="text/javascript">
var name =
  prompt("Пожалуйста, напишите свое имя","")
var d = new Date();
 var y =  d.getFullYear();
var m = d.getMonth() + 1;
var d = d.getDate();
var t =  m + '/' + d + '/' + y + ' ';
document.write("<TITLE>")
document.write
("Привет "+name+ ". Сегодня "
+t+ ". Спасибо, что зашли.");
document.write("</TITLE>")
</SCRIPT>
```

2.6. Концепция свойств

Концепция

Нам известно, что существуют объекты, например, document, и методы, например, write, которые воздействуют на объекты. Теперь рассмотрим концепцию свойств. Свойства представляют собой часть или качество объекта. Рассмотрим наиболее популярные из них и укажем, какую пользу они могут принести.

Сценарий

Ниже представлены несколько сценариев, где они составлены по одной схеме: для каждой команды объект.свойство

(object.property) создается переменная, затем переменные помещаются в команду document.write() для вывода.

Свойства объекта "navigator" (браузер)

```
<SCRIPT LANGUAGE="javascript">
var an = navigator.appName;
var av = navigator.appVersion;
var acn = navigator.appCodeName;
var ua = navigator.userAgent;
document.write("Вы пользуетесь <B>" +an+ "</B>,
версия " +av+ ".
<BR>Кодовое название " +acn+ ", заголовок "
+ua+ "." );  </SCRIPT>
```

Приведенный выше текст в скобках должен располагаться на одной строке.

Свойства объекта "document"

```
<SCRIPT LANGUAGE="javascript">
var bgc = document.bgColor;
var fgc = document.fgColor;
var lc = document.linkColor;
var al = document.alinkColor;
var vlc = document.vlinkColor;
var url = document.location;
var ref = document.referrer;
var t = document.title;
var lm = document.lastModified;
document.write("Цвет фона этой страницы <B>"
```

```
+bgc+ "</B>.")
document.write("<BR>Цвет текста этой страницы <B>" +fgc+
"</B>.")
document.write("<BR>Цвет ссылок этой страницы <B>" +lc+
"</B>.")
document.write("<BR>Цвет активной ссылки этой страницы
<B>" +al+ "</B>.")
document.write("<BR>Цвет посещенной ссылки этой страницы
<B>" +vlc+ "</B>.")
document.write("<BR>URL этой страницы <B>" +url+ "</B>.")
document.write("<BR>До этого вы были на странице <B>" +
ref+ "</B>.")
document.write("<BR>Заголовок этой страницы (TITLE) <B>"
+t+ "</B>.")
document.write("<BR>Последние изменения в документ
внесены: <B>" +lm+ "</B>.") </SCRIPT>
```

Свойства объекта "history"

```
<SCRIPT LANGUAGE="javascript">
var h = history.length;
document.write("До этой страницы вы посетили" +h+ "
страниц.") </SCRIPT>
```

Два свойства объекта "location" (адрес)

```
<SCRIPT LANGUAGE="javascript">
var hst = location.host
document.write("Страница находится на <B>" + hst +
"</B>." ) </SCRIPT>
```

<SCRIPT LANGUAGE="javascript">

var hstn = location.hostname

document.write("Страница находится на <B>" + hstn +

"</B>.") </SCRIPT>

Результат работы сценариев (возможный)

Некоторые сведения о вашем компьютере.

Вы пользуетесь Opera, версия 9.01 (Windows NT 5.1; U; ru).

Кодовое название Mozilla, заголовок Opera/9.01 (Windows NT 5.1; U; ru).

Цвет фона этой страницы #334775.

Цвет текста этой страницы #334775.

Цвет ссылок этой страницы #fbbcac.

Цвет активной ссылки этой страницы #d0550b.

Цвет посещенной ссылки этой страницы #fbbcac.

URL этой страницы http://jsp.newmail.ru/les7.htm.

До этого вы были на странице .

Заголовок этой страницы Концепция свойств.

Последние изменения внесены: Fri, 27 Jun 2003 00:09:36 GMT.

До этой страницы вы посетили 1 страниц.

Страница находится на www.mail.ru.

Страница находится на www.mail.ru.

Разбор сценария

В некоторых местах сценария, были добавлены команды <B> и </B> по обеим сторонам от имени переменной — внутри двойных кавычек. Из-за этого в некоторых местах шрифт ролучился жирный. Раз это оператор document.write, то в текст

можно вписать любые команды HTML, изменяющие текст. Только необходимо, чтобы команды HTML находились внутри двойных кавычек, чтобы они воспринимались как текст, иначе браузер посчитает их частью скрипта — это было бы ошибкой.

Свойства объекта navigator

```
<SCRIPT LANGUAGE="javascript">
var an = navigator.appName;
var av = navigator.appVersion;
var acn = navigator.appCodeName;
var ua = navigator.userAgent;
document.write("Вы пользуетесь <B>" +an+ "</B>,
версия " +av+ ".<BR>Кодовое название " +acn+ ",
заголовок " +ua+ "." ); </SCRIPT>
```

Текст в скобках должен быть весь на одной строке. Объект navigator имеет четыре свойства. appName сообщает название браузера, например, Netscape или Explorer. appVersion сообщает версию браузера и платформу, на которой он работает. appCodeName сообщает кодовое имя, данное браузеру, например, Netscape называет свой браузер Mozilla. userAgent сообщает версию используемого браузере.

Иногда важно знать версию браузера. Чуть позже мы изучим команды if (если). Зная браузер пользователя и его версию, можно дать команду: "Если браузер такой-то, сделать то-то."

Свойства объекта document

```
<SCRIPT LANGUAGE="javascript">
```

```
var bgc = document.bgColor;
var fgc = document.fgColor;
var lc = document.linkColor;
var al = document.alinkColor;
var vlc = document.vlinkColor;
var url = document.location;
var ref = document.referrer;
var t = document.title;
var lm = document.lastModified;
document.write("Цвет фона этой страницы <B>"
+bgc+ "</B>.")
document.write("<BR>Цвет текста этой страницы <B>" +fgc+
"</B>.")
document.write("<BR>Цвет ссылок этой страницы <B>" +lc+
"</B>.")
document.write("<BR>Цвет активной ссылки этой страницы
<B>" +al+ "</B>.")
document.write("<BR>Цвет посещенной ссылки этой страницы
<B>" +vlc+ "</B>.")
document.write("<BR>URL этой страницы <B>" +url+ "</B>.")
document.write("<BR>До этого вы были на странице <B>" +
ref+ "</B>.")
document.write("<BR>Заголовок этой страницы <B>" +t+
"</B>.")
document.write("<BR>Последние изменения в документ
внесены: <B>"
```

```
+lm+ "</B>.")
</SCRIPT>
```

Текст выше в скобках должен целиком располагаться в одной строке. Свойства документа HTML очень популярны в JavaScript. Здесь перечислены девять. На самом деле их тринадцать, но остальные четыре рассмотрим позже. Они перечислены ниже.

bgColor возвращает шестнадцатеричный код цвет фона.

fgColor возвращает шестнадцатеричный код цвета текста.

linkColor возвращает шестнадцатеричный код цвета ссылки.

alinkColor возвращает шестнадцатеричный код цвета активной ссылки.

vlinkColor возвращает шестнадцатеричный код цвета посещенной ссылки.

location возвращает URL страницы.

referrer сообщает, с какой страницы пришел пользователь. Если информация недоступна, то возвращается пустое место.

title возвращает заголовок документа HTML, т .е. текст между командами TITLE.

lastModified сообщает дату, когда были внесены последние изменения в страницу (на самом деле дату, когда страница была загружена на сервер или сохранена последний раз на жестком диске).

cookie (не показано) возвращает текстовый файл cookie.

anchors (не показано) возвращает количество анкеров HREF на странице.

forms (не показано) возвращает массив (список) объектов формы на странице.

links (не показано) возвращает количество всех отдельных ссылок.

В данном случае также с помощью команды if можно сказать: "Если время больше 6 вечера, пусть текст будет белый, а фон черный. Если еще нет 6 вечера, то пусть фон будет голубой, а текст зеленый". Существует множество способов использовать свойства документа.

Свойства объекта history

```
<SCRIPT LANGUAGE="javascript">
var h = history.length;
document.write("До этого вы посетили " +h+ " страниц.")
</SCRIPT>
```

Это очень популярный объект. Многие читатели хотят иметь возможность переместиться на одну или несколько страниц вперед или назад. Они пытаются воспроизвести кнопки "Вперед" и "Назад" на панели браузера. Объект history позволяет это сделать.

Объектом является журнал посещений history. Это список страниц, которые посетил браузер во время работы. Список истории позволяет реализовать кнопку "Назад" и просмотреть еще раз любую страницу.

Свойством является length (протяженность). Оно позволяет определить в сценарии количество страниц в папке "history".

Существует также метод go() (пойти), который позволяет передвигаться по history.length с указанным шагом.

Два свойства объекта location

```
<SCRIPT LANGUAGE="javascript">
var hst = location.host
document.write("Страница находится на <B>" + hst + "</B>.")
</SCRIPT>
<SCRIPT LANGUAGE="javascript">
var hstn = location.hostname
document.write("Страница находится на <B>" + hstn + "</B>.")
</SCRIPT>
```

Здесь объектом является location. Это URL на языке JavaScript, адрес страницы. Выше представлены два свойства объекта location: host, и hostname. Команды равноценны, так как выполняют одну и ту же задачу — сообщают URL в текстовом формате или адрес IP, в зависимости от сервера. Но... location.hostсообщает URL плюс "порт", с которым соединен пользователь. location.hostname сообщает только URL.

Если вы получаете одинаковый результат от обеих команд, значит, ваш сервер не соединил вас со специальным портом. Говоря техническим языком, свойство "порта" — null.

Эти две команды не работают, если просматривать страницу с жесткого диска. Результат может быть только в том случае, если она размещается на сервере, так как сценарию требуется URL для анализа.

Существует множество других свойств. Здесь даны общие представление о свойствах — как они используются и что делают наиболее часто используемые.

Задание

Используя одну из вышеприведенных команд типа объект.свойство, напишите сценарий JavaScript, который создает ссылку на страницу документа HTML на каком-либо сервере. Например, если вы находитесь на www.you.ru, сценарий JavaScript создаст ссылку на www.you.ru/index.html.

Также, какое бы свойство ни использовалось, присвойте ему имя переменной.

Имейте в виду, что страница должна находиться на сервере, на жестком диске сценарий не работает, так как там нет никакого location.host.

2.7. Иерархия объектов

Концепция

В JavaScript есть объекты, похожие на существительные или предметы. У объектов есть свойства, которые описывают их, как прилагательные описывают существительное. Мы ссылаемся на свойства с помощью схемы объект.свойство.

Еще у объектов есть методы, или действия, которые можно выполнить с объектом. Все методы имеют скобки и используются по схеме объект.метод(). У разных объектов имеются разные свойства и методы.

Познакомимся с иерархией объектов JavaScript. Что имеется в виду:

Window

Parent

Self

Location

Href

Document

Image

Src

Form

Text

Submit

Checkbox

Top

Frames

Результат действия иерархии

Все ссылки начинаются с наивысшего объекта, window (окно браузера), и идут по нисходящей. Окна и рамки (frames) принадлежат объекту window. На них не нужно ссылаться, если только их не больше одного. Top, self, parent и frames — "встроенные" имена для окон. Вот несколько примеров. Обратите внимание на иерархию.

document.mypic.src = "pic1.gif"

В самом начале window не требуется. Предполагается, что это все и так находится внутри окна. document.mypic.src указывает на изображение с именем mypic, и изменяет его содержимое на "pic1.gif". В данном случае document (документ) — это страница на которой находится элемент, mypic — имя элемента, а SRC — источник элемента ("pic1.gif").

document.write(location.href)

write() — это метод объекта document. location.href содержит полный URL окна. Обратите внимание, что location и document находятся на одном уровне. Это значит, что вы получаете адрес документа того же уровня.

Разбор иерархии объектов

Window

Parent

Self

Location

Href

Document

Image

Src

Form

Text

Submit

Checkbox

Top

Frames

Некоторые объекты также являются и свойствами.

window — только объект.

document является свойством окна, но в свою очередь и объектом.

form — это свойство документа, но также и объект со своими свойствами!

value (значение) и SRC (источник) — только свойства!

Здесь представлены не все объекты и свойства. Однако этого достаточно, чтобы понять концепцию в целом. Все ссылки начинаются сверху от window и идут по нисходящей. То есть, нельзя написать document.mytext.myform или mypic.src.document. Это неправильный порядок, следует писать слева направо от более общего к более конкретному.

Чтобы показать содержимое поля формы, необходимо использовать свойство value (значение), например, document.myform.mytext.value! Если написать просто document.myform.mytext, то будет получена информация о поле формы, но не о его содержании.

Value ("значение") является +некоторым показателем того, что нечто имеется или отсутствует в определенное время. Поле с флажком может иметь значение "on" или "off", в зависимости от того, задан он или нет. Текстовое поле может иметь значение "hidden" (скрытое), если не хотите, чтобы пользователь его видел. Текстовое поле, как указано выше, может содержать какую-то запись. Она будет значением этого поля.

Задание

Напишите полные ссылки, начиная с window, даже если это не требуется. Обратите внимание, что изображение имеет имя mypic, а форма — myform.

Ссылка на всю форму:

Ссылка на содержимое поля lname:

Ссылка на содержимое поля fname:

Замените изображение на "marigold.gif":

2.8. Создание функций

Концепция

При создании переменной результату команды или события JavaScript присваивается имя. При создании функции делают почти то же самое, только имя присваивается целой серии команд. Множество команд JavaScript объединяются в одну.

Сценарий

Сценарий состоит фактически из двух частей: собственно функции и команды onLoad, которая запускает функцию в работу.

Вот обе части:

```
<SCRIPT LANGUAGE="javascript">
<!-- Скрыть от браузеров, не поддерживающих JavaScript
function dateinbar()
{
var d = new Date();
var y = d.getFullYear();
var da = d.getDate();
var m = d.getMonth() + 1;
var t = da + '/' + m + '/' + y;
defaultStatus = "Вы прибыли на страницу " + t + ".";
}
// не скрывать --></SCRIPT>
```

...и команда onLoad в <BODY>:

```
<BODY BGCOLOR="xxxxxx" onLoad="dateinbar()">
```

Результат работы сценария

Результат работы сценария будет выводиться внизу в строке состояния.

Почти таким же сценарием мы пользовались для получения даты. Здесь нас интересует реализация сценария.

Разбор сценария

Здесь происходит две вещи. Сначала в первой части сценария создается функция. Потом команда, находящаяся внутри оператора HTML <BODY>, запускает работу функции. Разберем функцию.

```
function dateinbar()
{var d = new Date();
var y = d.getFullYear();
var da = d.getDate();
var m = d.getMonth() + 1;
var t = da + '/' + m + '/' + y;
defaultStatus = "Вы прибыли на страницу " + t + ".";}
```

Схема довольно понятная. Вы пишете function и задаете любое имя, точно так же, как мы делали с переменными.

Но обратите внимание, что после имени функции стоят круглые скобки, как после команды метода. То есть, при создании функции фактически создается новый метод для выполнения некоторой задачи.

Так же, как и имена переменных, имена функций могут быть любой длины, не содержать пробелов и не совпадать с другим словом, уже используемым в языке JavaScript. В данном случае мы выбрали имя dateinbar() (дата в строке состояния), потому что этофункция и делает — помещает дату в строку состояния.

При создании функции команды, которые составляют функцию, должны быть заключены в фигурные скобки {}. Они стоят сразу после имени функции и в самом конце.

Текст внутри фигурных скобок должен быть понятен.

создается переменная для года;

еще одна для числа;

еще одна для месяца;

затем четвертая для даты целиком.

Последняя команда новая:

defaultStatus = "Вы прибыли на страницу " + t + ".";

defaultStatus (строка состояния по умолчанию) — свойство объекта window. Его назначение — поместить текст в строку состояния внизу окна браузера.

Мы задействуем новый обработчик событий. Команда onLoad ("при загрузке") (обратите внимание на заглавные буквы) говорит браузеру, что, загружая страницу, он должен выполнить предписанное далее. В данном случае следует функция dateinbar().

Эта команда почти всегда располагается в строке <BODY> документа HTML. И почти всегда за ней следует функция, но это необязательно.

Расположение элементов имеет не последнее значение. Команда onLoad идет в строку BODY. Сценарий с функцией должен находиться в документе HTML между командами <HEAD> и </HEAD>. Хотя на самом деле его можно поместить где угодно, но если расположить его после команды onLoad, он

заработает только после того, как загрузится вся страница. Если сценарий располагается перед командой onLoad, то он загружается в память компьютера раньше, и когда onLoad вызовет его, он будет готов к работе.

Практически любой набор команд JavaScript можно записать в виде функции.

Задание

Создайте функцию, которая создает два запроса (prompt). (Подсказка: один следует за другим с новой строки.) Первый предлагает пользователю ввести свое имя, второй — фамилию. Затем та же функция должна вызвать окно сообщения (alert) с текстом:

Привет, "имя фамилия", добро пожаловать на "адрес страницы", мою замечательную страницу!

Можно немного усложнить, делая сценарий так, что слова "мою замечательную страницу" будут вставлены в текст команды alert не просто, а с помощью дополнительной переменной.

2.9. Команды последействия: onUnLoad и onMouseOut

Концепция

Это два последних обработчика событий, которые необходимо иметь в своем арсенале: onMouseOut и onUnload (обратите внимание на заглавные буквы). onMouseOver вызывает некое событие, если навести мышь, к примеру, на ссылку. В противоположность ей onMouseOut начинает действовать, если курсор увести со ссылки. Мы также знаем, что команда

onLoad запускает сценарий, когда страница загружается. Команда onUnload действует, когда пользователь уходит со страницы.

Сценарий

Следующий код использует события при перемещении указателя мыши:

<A HREF="index.htm" onMouseOver="window.status='Проходи мимо.';

return true"

onMouseOut="window.status='Так-то лучше, спасибо'; return true">

Наведите курсор на эту ссылку и сместите в сторону</A>

Использование команды onUnload при уходе со страницы:

<BODY onUnload="alert('Уже уходите?')">

Результат работы сценария

При размещении сценария на странице выводится текстовая ссылка "Наведите курсор на эту ссылку и сместите в сторону". Если навести курсор на ссылку и сместить в сторону несколько раз, то в строке состояния можно видеть изменяющиеся сообщения. Это первый результат. При нажатии на ссылку можно увидеть второй.

Разбор сценария

Эффекты с мышью, как легко видеть, создаются с помощью команд onMouseOver и onMouseOut.

Обратите внимание, что эти две команды никак не связаны между собой. Требуется, чтобы одно событие происходило, когда курсор мыши указывает на ссылку, а другое —

когда курсор мыши смещается со ссылки. Поэтому нужно писать их как две совершенно независимые команды, каждая из которых содержит свою команду return true.

Сообщение при уходе со страницы создается с помощью команды onUnload="alert('Уже уходите?')", которая добавлена в строку BODY документа HTML. Обратите внимание на двойные и одинарные кавычки. Внутри двойных — одинарные. Вторая двойная кавычка означает для браузера конец команды.

Задание

В этом задании необходимо использовать функции onUnload, onMouseOver, и onMouseOut. Сделайте следующее:

Создайте страницу с гипертекстовой ссылкой. Когда курсор указывает на ссылку, в строке состояния должны появляться слова: "Привет, пользователь 'название браузера'!". Когда курсор уходит со ссылки, в строке состояния должен появляться текст: "Не скучаете у нас на 'URL страницы'?" Если щелкнуть на ссылке, должно появиться окно со словами: "Уже уходите? Сейчас всего только 'текущее время'"; Время должно определяться с помощью функции. Не пользуйтесь командой onClick, чтобы вывести окно сообщения, возьмите команду onUnload.

2.10 Открываем новые окна

Концепция

Рассмотрим команды Javascript, которые используются для открытия нового окна. В новом окне будет выводиться другой документ HTML.

Сценарий

```
<SCRIPT type="text/javascript" >
window.open('opened.html', 'joe', config='height=300,width=300')
</SCRIPT>
```

Результат работы сценария

Представленный здесь сценарий будет только открывать окно. В этом окне выводится документ opened.html. Содержание документа будет рассмотрено позже.

Разбор сценария

Начнем с расположения сценария на странице. До сих пор всегда говорилось, что лучше помещать скрипты выше в документе, чтобы они быстрее загружались в память компьютера и начинали работать без задержки. Когда речь идет о функции, сценарий помещается между командами <HEAD>. Здесь будет сделано другое предложение.

Если собираетесь открывать новое окно, поместите команды, которые это делают, ближе к концу документа HTML. Проще говоря, пусть они идут в последнюю очередь. Причина простая: сначала загрузится страница, а потом откроется окно. Если команды стоят в начале, то окно появится прежде, чем пользователь увидит страницу. Скорее всего он закроет новое окно, прежде чем его можно будет использовать.

window.open

window (окно) — объект, а open (открыть) — метод, который на него воздействует. Теперь перейдем к конфигурации окна.

Конфигурация нового окна. В рассматриваемом случае мы имеем:

('opened.html', 'joe', config='height=300,width=300')

opened.html — это URL страницы, которая появится в новом окне. Если страница располагается на другом сервере, то необходимо добавить http:// и так далее.

joe — имя нового окна. Далее это будет важно.

config= указывает, что следующие команды относятся к конфигурации нового окна.

Команды config

Приведенные выше команды config открывают новое окно размером 300 на 300 пикселей.

Команды height (высота) и width (ширина) разделены только запятой без пробелов, а значения поставлены в одинарные кавычки, так как эти два элемента являются подкомандами config и должны выполняться совместно. Пробел для браузера означает конец команды. А это ошибка.

Для команды config существует множество подкоманд. Про высоту (height) и ширину (width) мы уже знаем, они определяются в пикселях. Остальные подкоманды употребляются со словами "yes" или "no" в зависимости от того, нужны ли в новом окне эти элементы. (Можно ставить "1" вместо "yes" и "0" вместо "no", но это не обязательно.)

Никаких пробелов между подкомандами не должно быть и использовать надо одинарные кавычки. Пробел равносилен ошибке.

toolbar= отвечает за наличие панели инструментов во вновь открытом окне. Панель инструментов в верхней части окна браузера содержит кнопки НАЗАД, ВПЕРЕД, СТОП и т.д. ()

menubar= отвечает за наличие строки меню с элементами ФАЙЛ, ПРАВКА, ВИД и т.д.

scrollbars= отвечает за наличие полосы прокрутки.

resizable= указывает, может ли пользователь изменить размер окна по своему желанию.

location= отвечает за наличие адресной строки во вновь открытом окне, в которой выводится URL страницы.

directories= отвечает за наличие строки каталогов в новом окне, которая содержит закладки и т.п.

В открывающееся новое окно загружается документ HTML opened.html. Это обычная страница HTML, которая может содержать любые команды, в частности, с этой страницы можно управлять загрузкой других документов. Например, чтобы открыть главную страницу INDEX в основном окне, надо поместить на ней следующий код:

<A HREF="http://www.index.ru" TARGET="main window"></A>

Основное окно всегда имеет по умолчанию имя "main". Поэтому в команду HREF документа HTML добавляется просто команда TARGET="—" с указанием main для окна, в которое должна загрузиться страница.

А если надо, чтобы страница загрузилась в новом окне, необходимо написать команду ссылки HREF с указанием окна joe.

Можно открыть на самом деле несколько окон, добавляя несколько команд window.open. Надо только задать окнам различные имена. Можно создавать также ссылки между окнами, указывая необходимые имена окон.

Можно создать также в документе ссылку, которая будет закрывать окно. Вот как это делается:

<A HREF="" onClick="self.close()">Щелкните, чтобы закрыть</A>

Это обычная ссылка HREF, которая никуда не ведет. Задание ссылки таким образом позволяет избежать загрузки страницы. Закрывает окно команда onClick="self.close()".

self (само, себя) — это свойство может относиться к любому объекту. В этом случае это свойство окна. Команда close (закрыть) закрывает окно.

Допустим, что требуется открыть окно по команде, а не когда пользователь заходит на страницу. Это можно сделать следующим образом:

<A HREF="les11.htm" onClick="window.open('opened.html', 'joe',

config='height=300,width=300')">Щелкните, чтобы открыть 'joe'</A>

Это ссылка HREF, которая направлена на саму себя. Команда onClick делает работу, а параметры содержатся в скобках ().

Задание

Напишите сценарий, который откроет новое окно со всеми характеристиками. Пусть оно будет размером 300 на 500 пикселей и содержит две ссылки:

одна откроет новую страницу в главном окне; вторая откроет новую страницу в том же окне.

Страница, которая откроется в том же маленьком окне, должна содержать ссылку, закрывающую окно.

Сделайте фон страницы желтым (ffff00).

2.11 Открытие окна с помощью функции

Концепция

Создадим функцию, которая откроет новое окно, — причем и новое окно, и все его содержимое будет содержаться в том же документе HTML. То есть, мы вложим две страницы в одну.

Сценарий

```
<SCRIPT type="text/javascript">
function openindex()
  { var OpenWindow=window.open("", "newwin", "height=300,width=300");
OpenWindow.document.write("<HTML>");
OpenWindow.document.write("<TITLE>Новое окно</TITLE>");
OpenWindow.document.write("<BODY BGCOLOR='00ffff'>");
OpenWindow.document.write("<CENTER>");
OpenWindow.document.write("<font size=+1>Новое окно</font><P>");
OpenWindow.document.write("<a href='http://www.intuit.ru' target='main'>
```

Эта ссылка
 откроется в основном окне<p>");

OpenWindow.document.write("<P><HR WIDTH='60%'><P>");

OpenWindow.document.write("

Эта ссылка закроет окно<p>");

OpenWindow.document.write("</CENTER>");

OpenWindow.document.write("</HTML>");}

</SCRIPT>

...и в команде BODY:

onLoad="openindex()"

Результат работы сценария

Результат такой же, что и в прошлый раз: открылось окно того же размера с теми же двумя ссылками. Разница в том, что все это было написано на одной странице.

Разбор сценария

Главная часть сценария, содержащая функцию, помещается между тегами <HEAD> и </HEAD>, как большинство функций.

По обычной схеме для функции задается имя openindex(). Затем следуют фигурные скобки. Теперь подходим к основному моменту. Создаем переменную OpenWindow, под которой скрывается команда window.open(). Она выглядит следующим образом:

var OpenWindow=window.open("", "newwin", "height=300,width=300");

Формат тот же. Единственная разница в том, что не указан URL. Пустые парные кавычки говорят браузеру, что он должен искать в сценарии информацию о новом окне, — точно так

же, как и в случае отсутствия URL в команде, которая закрывает окно. Оно бы не закрылось, если бы начала загружаться новая страница. То же самое и тут. Браузер стал бы загружать новую страницу, а не выполнять сценарий.

Теперь начинаем создавать страницу HTML, которая будет в новом окне. Вот первая строка текста:

OpenWindow.document.write("<HTML>")

Команда говорит, что строка текста должна быть записана в документ переменной OpenWindow (новое окно) .

Каждая новая строка следует той же схеме. Можно написать сотню строк, создающих законченную страницу. Наконец обработчик событий onLoad в команде BODY вызывает функцию.

Задание

Написать функцию, которая открывает окно. Документ, который появится в окне должен иметь зеленый фон и заголовок TITLE: "Привет, "имя пользователя", вот твое окно!" Имя пользователя можно узнать с помощью запроса (prompt). Надо добавить еще ссылку, которая закроет окно.

2.12 Метод 'Confirm' (Введение в if и else)

Концепция

Команда confirm (подтвердить) действует очень похоже на метод alert, но добавляет в диалоговое окно кнопку "Отмена" (Cancel). И то, и другое — методы.

Одна команда сама по себе многого не дает. Нет никакой разницы, что выбирать — "ОК" или "ОТМЕНА". Но стоит добавить

функции if (если) и else (иначе), и можно создать интересные эффекты.

Сценарий

Прежде всего посмотрим на базовый формат:

```
<SCRIPT type="text/javascript">
confirm("Уверены, что хотите войти?")
</SCRIPT>
```

Выглядит знакомо. То же самое, что и alert, кроме слова confirm. Как видите, сценарий делает не очень много. Но вот та же команда с некоторыми добавлениями:

```
<SCRIPT type="text/javascript">
if (confirm("Уверены, что хотите посетить INTUIT?") )
{parent.location='http://www.index.ru/';
alert("Счастливого пути");}
else
{alert("Тогда оставайтесь");}
</SCRIPT>
```

Результат работы сценария

Выводится ссылка с вопросом. Только на этот раз, если нажать "OK", то произойдет переход по ссылке, а если щелкнуть на "Отмена", то останетесь на странице.

Разбор сценария

Сценарий говорит:

if (confirm("Уверены, что хотите посетить INDEX?"))

Это значит Если (Здесь можно сделать выбор).

В этом случае confirm предлагает варианты: "OK" или "Отмена". Можно считать их ответами Да и Нет. Следует обратить внимание на скобки. После команды if всегда идут круглые скобки, но, как известно, команда confirm тоже требует скобок. Следовательно, берем две пары скобок, одна внутри другой.

Сразу же после этого идут команды, выполняемые при каждом варианте ответа. Обратите внимание на фигурные скобки {}. Зачем? Потому что в действительности это функции. Первая из них показывает, что должно произойти, если пользователь выберет ОК (или Да).

```
{parent.location='http://www.index.ru/';
alert("Счастливого пути");}
```

Известно, что parent.location является командой, создающей ссылку. Дальше идет обыкновенная команда alert.

Если выбрать ОК, то выполнится функция, следующая непосредственно за оператором if (если). "Cancel" (Отмена) — другой выбор. Сразу после фигурной скобки идет команда else (иначе), как бы "если нет". И тогда следующий текст...

```
else{
alert("Тогда оставайтесь");}
```

...означает: если нет, тогда послать сообщение и не менять страницу.

Все это вместе и дает пользователю возможность выбора: входить или не входить.

Это самые основы использования if и else.

Задание

Преобразуйте рассмотренный выше сценарий в функцию. И сделайте так, чтобы при отмене (Cancel), кроме окна, еще появлялась какая-нибудь надпись в строке состояния.

Можно также попробовать сделать так, чтобы при выборе ОК страница перехода открывалась в новом окне.

2.13. Математические вычисления

Концепция

Рассмотрим как производить математические вычисления с помощью JavaScript.

Сценарий

```
<BODY>
  <SCRIPT type="text/javascript">
     var numsums = 10 + 2
         alert("10 + 2 равно " + numsums)
     var x = 10
         alert("десять — это " + x)
     var y = x * 2
         alert("10 X 2 = " + y)
     var z = "Привет " + "Пока"
     alert(z)
  </SCRIPT>
</BODY>
```

Результат работы сценария

При выполнении сценарий последовательно выводит ряд окон, содержащих:

10 + 2 равно 12

десять — это 10

10 X 2 = 20

Привет Пока

Разбор сценария

```
<BODY>
  <SCRIPT type="text/javascript">
     var numsums = 10 + 2
         alert("10 + 2 равно " + numsums)
     var x = 10
         alert("десять — это " + x)
     var y = x * 2
         alert("10 X 2 = " + y)
     var z = "Привет " + "Пока"
     alert(z)
  </SCRIPT>
</BODY>
```

В сценарии задается переменная numsums. Она равна 12 (10+2). Затем эта переменная используется в методе alert и выводит "10 + 2 = переменная" или 12.

Другая переменная, x, задается равной 10. Затем метод alert выводит это значение.

Следующая переменная, y, равна переменной x, умноженной на 2. Этот результат затем выводится в окне alert.

Затем создается переменная z, которая показывает, что можно соединять текст с помощью знака сложения. Эта переменная выводится с помощью метода alert.

Основные моменты:

Переменные начинаются со слова VAR (от variable, переменная), затем идет имя, знак = и значение переменной. Имя переменной может состоять из одного или нескольких символов (буквы, цифры, символ подчеркивания). Но лучше использовать содержательные имена.

Имена переменных различают регистр! То есть, Xvar и xvar — это два разных имени переменных.

Допустимая длина имени переменной существенно различна для разных браузеров. В целях безопасности следует брать имена не больше 10 символов. Не используйте в именах пробелы.

Значение, присваиваемое текстовой переменной, ставится в кавычки. Числовые переменные не ставятся в кавычки, иначе сценарий воспримет их как текст с числовым значением 0!

Операции сложения, вычитания, умножения и деления обозначаются знаками +, -, *, и / соответственно.

Знак плюс (+) выполняет две задачи: сложение чисел или соединение вместе двух строк текста (например, "Joe" + " Burns" будет "Joe Burns").

Все языки программирования имеют зарезервированные слова, например, названия команд. Использование зарезервированных слов в качестве имен переменных будет приводить к ошибкам.

Если необходимо, можно применять в именах переменных вместо пробела знак подчеркивания _ user_name.

Задание

Перепишите приведенный выше код JavaScript в виде функции. Можно, при желании, изменить некоторые математические операции, например, на деление. Включите в тело документа HTML приветственное сообщение. Используйте для выполнения функции событие onLoad.

2.14 Изменение изображения с помощью события onMouseOver

Концепция

В данном примере будут рассмотрены дополнительные возможности использования событий onMouseOver и onMouseOut. Как мы говорили ранее, любое событие может запускать на выполнение функцию или оператор JavaScript. Вспомните команду onLoad в теле документа HTML, которая вызывает код JavaScript из заголовка HEAD.

Представленные здесь два события происходят, когда указатель мыши перемещается на ссылку или смещается со ссылки. Ранее эти обработчики событий использовались для создания эффекта появления текста в строке состояния.

Обратите внимание на то, что теги <SCRIPT> и </SCRIPT> не требуются. Обработчики событий onMouseOver и onMouseOut встраиваются в тег HTML <A HREF>. Также отметим, что для отключения вывода рамки вокруг изображения в теге <IMG SRC> включен атрибут BORDER="0".

Сценарий

```
<A HREF="http://www.index.ru"
onMouseOver="document.pic1.src='on.gif'"
onMouseOut="document.pic1.src='off.gif'">
<IMG SRC="off.gif" BORDER=0 NAME="pic1">
</a>
```

Результат работы сценария

На странице выводится изображение off.gif. Если навести на изображение указатель мыши, то изображение изменится на on.gif. При смещении указателя мыши с изображения возвращается изображение off.gif.

Разбор сценария

```
<A HREF="http://www.intuit.ru"
onMouseOver="document.pic1.src='on.gif'"
onMouseOut="document.pic1.src='off.gif'">
<IMG SRC="off.gif" BORDER=0 NAME="pic1">
</a>
```

Так как обработчики события были достаточно хорошо рассмотрены ранее, то этот сценарий не представляет трудностей. Когда курсор уходит с изображения, появляется off.gif. Когда указывает на изображение, появляется on.gif.

Обратите внимание, что команда IMG связана с обработчиками onMouse в команде HREF через команду NAME ="pic1" . Это необходимо для связи команд.

Основные моменты: onMouseOver и onMouseOut различают регистр. Нельзя менять заглавные и строчные буквы. Так как

необходимо ставить кавычки после onMouseOver= и onMouseOut=, то название файла *.gif берется в одинарные кавычки, а не в двойные. document.pic1.src следует иерархии объектов, document относится к текущему объекту документа HTML, а pic1 — это имя объекта изображение (имя можно придумать какое угодно). src (источник) — это свойство объекта изображение, которое указывает файл изображения.

В этом примере onMouseOver меняет источник изображения на on.gif. OnMouseOut заставляет объект изображение вывести off.gif. Для наилучшего эффекта картинки должны быть одинакового размера.

Задание

Создайте страницу HTML. Разместите все по центру страницы. Используйте тег H1 со своим именем. Под ним поместите изображение Bubble1.gif. Когда курсор мыши укажет на это изображение, оно должно измениться на изображение Bubble2.gif. Когда курсор мыши сместится с этой ссылки, снова должно появиться изображение Bubble1.gif.

2.15 Изменение изображения с помощью функции

Концепция

Вот еще один пример использования onMouseOver и onMouseOut. На этот раз вместо включения оператора JavaScript для смены картинки в тег <A HREF> обработчики событий onMouseOver и onMouseOut вызывают функцию.

В общем, когда нужна только одна команда JavaScript, можно включить ее в тег HTML <A HREF>. Для нескольких операторов

JavaScript больше подходит функция. В реальном мире на странице часто требуется многократное изменение изображения с помощью JavaScript.

Сценарий

```
<HTML>
<HEAD>
  <title> Пример JavaScript </title>
<SCRIPT type="text/javascript">
function up()
{     document.mypic.src="up.gif" }
function down()
{     document.mypic.src="down.gif"  }
</SCRIPT>
</HEAD>
  <BODY>
    <CENTER>
    <h2>Пример анимации</h2>
<A HREF="http://www.intuit.ru"
  onMouseOver="up()"  onMouseOut="down()">
<IMG SRC="down.gif" NAME="mypic" BORDER=0></A>
  </BODY>
</HTML>
```

Результат работы сценария

Пример анимации с помощью OnMouseOver и OnMouseOut.

Если быстро двигать курсором мыши на изображении, то возникает ощущение ожившего изображения — анимации.

Разбор сценария

Обратите внимание, что в сценарии созданы две функции.

```
<SCRIPT type="text/javascript">
function up()
{ document.mypic.src="up.gif" }
function down()
{ document.mypic.src="down.gif" }
```

Функции выполняют то же, что и команды в предыдущем сценарий. Сначала документ, потом имя, присвоенное объекту, и наконец SRC. Функции названы up() и down().

Теперь посмотрим на вызов функции:

```
<A HREF="http://www.intuit.ru"
onMouseOver="up()" onMouseOut="down()">
<IMG SRC="down.gif" NAME="mypic"
BORDER=0></A>
```

Этот пример имеет один оператор JavaScript, хотя обычно функции используются для объединения нескольких операторов. Если вы решите использовать многократную смену картинок с помощью JavaScript, помните, что для каждого случая нужно создавать имя новой функции и соединять определенные изображения с этими функциями, изменяя также атрибут NAME. Например: вы хотите поместить на страницу еще одну такую же меняющуюся картинку. Для этого создаем две новые функции, копируя предыдущие две функции и добавляя к именам цифру 2 (обратите на это внимание в коде ниже). Затем необходимо изменить в каждом случае атрибут NAME, поэтому изменяем имя

на mypic2. Получаем примерно следующее в разделе заголовка HEAD:

```
<SCRIPT LANGUAGE="JavaScript">
function up()
{   document.mypic.src="up.gif"}
function down()
{    document.mypic.src="down.gif"  }
function up2()
{    document.mypic2.src="up.gif" }
function down2()
{    document.mypic2.src="down.gif"  }
</SCRIPT>
```

... и примерно следующее для вызова двух различных изображений:

```
<A HREF="http://www.index.ru"
        onMouseOver="up()"
        onMouseOut="down()">
    <IMG SRC="down.gif" NAME="mypic"
BORDER=0></A>
<a href="http://www.index.ru"
        onMouseOver="up2()"
        onMouseOut="down2()">
    <IMG SRC="down.gif" NAME="mypic2"
BORDER=0></A>
```

При добавлении новой меняющейся картинки, следует давать новые имена. Если требуется три случая, то добавьте цифру 3 везде, где выше добавлена 2. Если четыре, добавьте 4, и т.д.

Задание

Переделайте последнее задание, использующее Bubble1.gif и Bubble2.gif. Создайте две функции для смены этих изображений.

2.16. Вызов функции в формы

Концепция

Рассматриваемый сценарий использует форму для выбора цвета фона, голубого или розового. Выбор цвета делается с помощью кнопок формы.

Формы всегда начинаются тегом <FORM> и заканчиваются тегом </FORM>.

Сценарий

Пример ниже снова показывает весь документ HTML:

```
<html>
<head>
<SCRIPT type="text/javascript">
function newcolor(color)
{  alert("Вы выбрали " + color)
  document.bgColor=color}
</SCRIPT>
</HEAD>
<BODY>
<p>Выберите цвет фона</p>
<FORM>
```

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

Результат работы сценария

На странице выводятся две кнопки с надписями "Голубой" и "Розовый". При нажатии на любую кнопку цвет фона меняется в соответствии с указанным на кнопке цветом.

Разбор сценария

Литерал является значением (VALUE), которое не изменяется. Литерал может быть числом, именем или любой случайной последовательностью чисел и имен. Помните только о том, что литерал невозможно изменить.

Строка является любой последовательностью букв или цифр в одиночных или двойных кавычках. Таким образом следующий фрагмент сценария:

```
onClick="newcolor('lightblue')"
```

... определяет строку литерал "lightblue". Все понятно? Отлично.

Вот снова сценарий и элементы ввода:

```
function newcolor(color)
{  alert("Вы выбрали " + color)
```

```
 document.bgColor=color}
<FORM>
 <INPUT TYPE="button" VALUE="Голубой"
    onClick="newcolor('lightblue')">
 <INPUT TYPE="button" VALUE="Розовый"
    onClick="newcolor('pink')">
</FORM>
```

Обратите внимание, мы передаем в функцию newcolor() (новый цвет) строку литерал, 'lightblue' или 'pink'. Она находится в одинарных кавычках, потому что имя функции стоит в двойных.

Когда вы нажимаете кнопку, строка в скобках ('pink' или 'lightblue') передается в функцию newcolor().

Функция ждет, пока поступит необходимая ей информация. Вспомните, что во всех функциях до сих пор скобки были пустые. Функции имели 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 свойству document.bgColor. Получив все данные, функция вступает в действие: всплывает окно и меняется цвет фона.

Атрибут VALUE (значение) в команде INPUT не является свойством JavaScript, он выводит текст на кнопку. Он не влияет на свойства JavaScript.

Задание

Перепишите скрипт так, чтобы, открываясь, страница предлагала пользователю ввести имя. При выборе цвета должно всплывать окно со словами "Эй, (имя)! Вы выбрали (цвет)...".

2.17 Поля формы и свойство value

Концепция

Переделаем данные, которые пользователь введет в поле формы. Затем эти данные будут использованы для поиска в Yahoo.

Сценарий

```
<SCRIPT>
function Gofindit(){
 var searchfor = document.formsearch.findthis.value;
{
 var FullSearchUrl =
"http://www.yahoo.com/bin/query?p=" + searchfor ;
location.href = FullSearchUrl;}}
</SCRIPT>
<FORM NAME="formsearch" action="">
Найдите в Yahoo:
<INPUT NAME="findthis" SIZE="40" TYPE="text">
<INPUT TYPE="button" VALUE="Искать"
onClick="Gofindit()">
</FORM>
```

Результат работы сценария

На странице выводится строка "Найдите в Yahoo:", поле ввода и кнопка с надписью "Искать".

Разбор сценария

Этот сценарий требует четкого понимания иерархии объектов.

Создаем функцию с переменной searchfor (искать) под названием formsearch, внутри элемента findthis (найти), который обладает свойством value (значение). Она будет результатом чего-то происходящего в объекте document.

Вторую функцию помещаем внутри первой. Вторая пара {фигурных скобок}.

Для второй функции создаем еще одну переменную FullSearchUrl, которая представляет собой адрес поисковой машины Yahoo плюс значение переменной searchfor, полученное через команду document.formsearch.find.value.

Наконец, location.href приравнивается переменной FullSearchUrl. После выполнения функции пользователь попадет на итоговую страницу поиска.

Теперь переходим к командам формы. Их две: текстовое поле (TEXT), куда пользователь вводит свой запрос, и кнопка, запускающая функцию. Форма в целом называется formsearch. Затем для текстового поля задаем имя findthis. Дальше соединяем кнопку с командой onClick, которая запускает функцию.

Заканчиваем форму командой </FORM>.

Задание

Измените сценарий так, чтобы он вызывал другую поисковую систему. И еще, пусть при отправке запроса появляется окошко с надписью "Сейчас поищем..."

2.18 Передача данных в функцию

Концепция

Обычно JavaScript в соединении с формами используется для проверки ввода данных.

Сценарий

Здесь снова представлен весь документ HTML, чтобы показать размещение отдельных элементов.

```
<HTML>
<HEAD>
<SCRIPT type="text/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("Длина имени "
+ document.myform.fname.value.length)}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform" action="">
  Ваше имя?
```

```
 <INPUT TYPE="text" NAME="fname"><p>
Ваша фамилия?
 <INPUT TYPE="text" NAME="lname"><p>
 <INPUT TYPE="button"
   VALUE="Отправить"
      onClick="doit()">
</FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится два поля ввода с просьбой ввести имя и фамилию и кнопка "Отправить". После нажатия на кнопку введенные данные отображаются в последовательно появляющихся окнах alert.

Разбор сценария

Начнем с элементов формы:

```
<FORM NAME="myform" action="">
Ваше имя:
   <INPUT TYPE="text" NAME="fname"><p>
Ваша фамилия:
   <INPUT TYPE="text" NAME="lname"><p>
<INPUT TYPE="button" VALUE="Отправить" onClick="doit()">
</FORM>
```

Мы дали форме имя myform. Текстовое 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. Тому, что написано в поле fname, будет присвоено имя fname.

Когда пользователь нажмет на кнопку (обработчик события onClick), выполнится функция doit().

Теперь посмотрим на функцию:

```
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("Длина имени "
+ document.myform.fname.value.length)}
```

В скобках функции doit() ничего нет. Но по иерархическим командам понятно, что функция использует данные, введенные в форму.

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

Дальше идет переменная greeting (приветствие). Содержимое greeting выводится с помощью команды alert(greeting). Когда пользователь нажимает на кнопку, всплывает окно с его именем.

Второе окно включает в себя переменную greeting. Появляется надпись: "Привет, (имя) (фамилия)", которая составлена с помощью данных, полученных через форму. Третье окно с неким текстом вызывает команду: document.myform.fname.value.length. Эта команда выводит длину (length) слова, введенного в поле формы fname. Если fname содержит имя "Коля", то длина равна 4. Команда length следует за value. Таким образом она подсчитает количество букв в тексте. length — это тоже свойство.

Задание

Составьте документ HTML с формой aform. В ней должно быть два текстовых поля (одно для города, другое для страны) и кнопка. Напишите функцию с переменной, которая содержит слова "Мне нравится ". Когда пользователь нажмет на кнопку, должно всплывать окно со следующей надписью:

Мне нравится город (страна).

(по результатам тех данных, которые пользователь вводит в форму)

Покажите длину (length) названия города.

2.19 Создание случайных чисел

Концепция

В этом примере рассматриваются случайные числа. Для генерации случайных чисел JavaScript использует дату и время.

Число после знака % является ограничивающим числом. Пример ниже выбирает случайное число между 1 и 10.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function rand()
    {       var now=new Date()
       var num=(now.getSeconds())%10
       var num=num+1
       alert(num)     }
</SCRIPT>
</HEAD>
<BODY>
 <h1>Случайные числа</h1>
 <form>
   <INPUT TYPE="button"
   VALUE="Случайные числа от 1 до 10"
   onClick="rand()">
 </FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится кнопка с надписью "Случайные числа от 1 до 10", при нажатии на которую появляется окно со случайным числом от 1 до 10.

Разбор сценария

Начнем в этот раз с функции:

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

Выбор случайного числа осуществляется в несколько шагов. Создаем функцию. Наша называется rand(). Потом создаем переменную для метода new Date().

Создаем еще одну переменную, num. Она содержит метод getSeconds(), так как в данном случае мы используем секунды для выбора случайного числа.

JavaScript, как и многие другие компьютерные языки, начинает отсчет с нуля. Поэтому элемент %10 говорит JavaScript, что случайное число будет выбираться из чисел от 0 до 9.

Оператор % возвращает остаток от деления. Предположим, что функция getSeconds() вернула значение 20 секунд. При делении на 10 получаем остаток 0. Сценарий возвращает 0. Пусть число секунд равно 12. Остаток при делении на 10 будет равен 2.

Прибавляя к случайному числу единицу (num=num+1), мы получаем числа не от 0 до 9, а от 1 до 10.

Наконец alert выводит число. Теперь элемент, запускающий функцию:

```
<form action="">
  <INPUT TYPE="button"
```

```
  VALUE="Случайное число от 1 до 10"
  onClick="rand()">
</form>
```

Это обычная кнопка, которая запускает выполнение приведенной выше функции.

Задание

Напишите программу JavaScript, в которой пользователь нажимал бы кнопку в форме, а программа выводила бы случайное число от 0 до 4 со словами: "Ваше случайное число: "x".

2.20 Оператор if и ветвление

Концепция

Этот пример знакомит с оператором IF (если), который позволяет решить, что делать, "если" выполняется какое-то условие. Программа спрашивает пользователя, любит ли он спорт. Если он отвечает "да", то программа отвечает "Я тоже люблю спорт". Если пользователь говорит "нет", то программа отвечает"Я тоже ненавижу спорт". Это немного глуповато, но для краткого знакомства вполне подходит.

Отметим, что если пользователь вводит что-то отличное от "да" или "нет", то программа выводит сообщение "Отвечайте да или нет".

За оператором IF следует условие и указание, что делать, если оно верно. Можно проверять одно условие или несколько. Программа знает, где начинаются и кончаются исполняемые после условия операторы, потому что они заключены в {фигурные скобки}.

Сценарий

```
<HTML>
<HEAD>
 <SCRIPT type="text/javascript">
  function askuser()
  {   var answer="   "
   var statement="Отвечайте да или нет"
   var answer=prompt("Вы любите спорт?")
   if ( answer == "да")
    {statement="Я тоже люблю спорт!"}
   if(answer == "нет")
    {statement="Я тоже ненавижу спорт!"}
   alert(statement)    }
  </SCRIPT>
</HEAD>
<BODY>
 <h1>Деятельность</h1>
  <FORM action="">
   <INPUT TYPE="button" VALUE="Нажми здесь!"
     onClick="askuser()">
  </FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится кнопка со словами "Нажми здесь!", при нажатии на которой появляется окно с вопросом и полем ввода. В зависимости от ответа выводятся различные сообщения.

Разбор сценария

Начнем с кнопки:

```
<FORM>
<INPUT TYPE="button" VALUE="Нажми здесь!" onClick="askuser()">
</FORM>
```

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

Фрагмент сценария с функцией:

```
function askuser()
 {  var answer="   "
  var statement="Отвечайте да или нет"
  var answer=prompt("Вы любите спорт?")
  if ( answer == "да")
   {statement="Я тоже люблю спорт!"}
  if(answer == "нет")
   {statement="Я тоже ненавижу спорт!"}
  alert(statement) }
```

Значение переменной answer (ответ) равно трем пробелам. Это пустое пространство будет заполнено тем ответом, который пользователь введет в поле ввода.

На те случаи, когда пользователь не отвечает "да" или "нет", создается переменная statement. Скоро будет понятно, зачем это нужно.

Затем переменная answer задается как результат запроса prompt. Теперь у нас две переменные под одним именем. Помните об этом.

Следом за if идет условие в (круглых скобках). Не забывайте о них.

В условии, которое состоит из того, что находится между скобок, используется не знак =, а знак == (два знака равенства)! Одинарный знак = используется вне скобок.

Помните, строковые значения должны заключаться в кавычки. "да" и "нет" являются строками текста.

Процесс происходит следующим образом:

prompt предлагает ответить на вопрос;

проверяется введенное значение;

если ответ "да", появляется окно со словами: "Я тоже люблю спорт!"

если ответ "нет", появляется окно со словами: "Я тоже ненавижу спорт!"

если ответ ни тот, ни другой, переменная answer остается пустой и строка "Отвечайте да или нет" посылается в alert.

Следует иметь ввиду что JavaScript различает регистр символов. То есть, если написать "НЕТ" или "Нет", условие не будет выполнено! Чтобы условие было

верно, необходимо ввести "нет". Исправить это можно, добавив еще несколько условий IF .

Задание

Перепишите программу так, чтобы она спрашивала пол пользователя. 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. Например:

if (answer == "M")

Если ввести "м", то условие не будет выполняться!

2.21 Операторы if/else

Концепция

Этот второй пример с if включает случайное число, две функции и знакомство с If / Else.

Операторы If / Else (если/иначе) предоставляют дополнительный контроль над программой, позволяя принимать решениие в обоих случаях: и когда условие выполнено, и когда не выполнено.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/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.");      }
  </SCRIPT>
</HEAD>
<BODY onLoad="rand()">
  <h1>Я загадал число от 1 до 10</h1>
  <FORM NAME="myform">
    <INPUT TYPE="button" VALUE="Угадай"
       NAME="b1" onClick="guessnum()">
  </FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится текст "Я загадал число от 1 до 10" и кнопка с предложением "Угадай". При нажатии на кнопку выводится окно с полем ввода и словами "Угадай, какое". В зависимости от введенного числа пользователь получит то или иное сообщение.

Разбор сценария

Начнем с команды BODY.

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

На этот раз функция запускается не событием onClick в описании кнопки, а событием onLoad в команде BODY. В этом случае к тому времени, когда пользователь нажмет кнопку, число уже будет выбрано. Если сделать это по-другому, то каждый раз, нажимая на кнопку, вы будете получать новое случайное число. А оно должно оставаться одним и тем же, пока вы пытаетесь угадать.

Теперь первая функция:

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

Функция выбирает наугад число из даты и времени от 0 до 9 и присваивает его num. Потом прибавляет к num единицу, чтобы выбор осуществлялся между 1 и 10. Мы делали это в прошлом уроке.

Теперь вторая функция:

```
function guessnum()
  {guess=prompt("Угадай, какое?")
   if (eval(guess) == num)
   {alert("ПРАВИЛЬНО!!!");
    rand();    }
     else     alert("Нет. Попробуй еще раз.");     }
```

В памяти компьютера уже есть число, полученное через первую функцию. Вторая дает возможность угадать его. Смотрите, что происходит:

С помощью запроса prompt 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.

Переходим к IF/Else. Если (if) guess (догадка) равна загаданному числу num, тогда запускается команда alert ("ПРАВИЛЬНО").

Если это не так (else), тогда запускается другая команда alert.

Остальное уже знакомо:

<form name="myform" action="">

<input type="button" value="Угадай" name="b1" onClick="guessnum()">

</form>

Кнопка запускает функцию, которая дает возможность угадать задуманное число.

Задание

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

В этом случае возможны только три результата: слишком много, слишком мало или правильно.

2.22 Случайный выбор фраз и изображений

Концепция

Рассмотрим еще один пример использования IF. Важно хорошо овладеть этой техникой ветвления, которая позволяет создавать более оригинальные и более интерактивные программы.

Сценарий

```
<HTML>
<BODY>
<SCRIPT type="text/javascript">
  var0="От пирогов не толстеют"
  var1="Кто ходит в гости по утрам"
  var2="До пятницы я совершенно свободен"
 now=new Date()
 num=(now.getSeconds() )%3
  if (num == 0)
   {cliche=var0}
  if (num == 1)
   {cliche=var1}
  if (num == 2)
    {cliche=var2}
 document.write(cliche + "<br>")
 document.write("Случайное число: "
   + num)
</SCRIPT>
<p>.... как я обычно говорю.
</BODY>
</HTML>
```

Результат работы сценария

Если страницу с этим сценарием перезагрузить несколько раз, то случайным образом будет выводиться одна из фраз и случайное число 0, 1 или 2.

Разбор сценария

Начнем со случайного числа:

```
var0="От пирогов не толстеют"
  var1="Кто ходит в гости по утрам"
  var2="До пятницы я совершенно свободен"
 now=new Date()
 num=(now.getSeconds() )%3
document.write("Случайное число: "     + num)
```

Оператор document.write должен располагаться на одной строке.

Мы создали три переменные. Это неизменяемые фрагменты текста и потому заключены в двойные кавычки.

Следующий шаг: программа создает случайное число с помощью часов компьютера. %3 указывает на то, что будет выбрана цифра из 0, 1 и 2. На этот раз мы не прибавляем к num единицу, так как нам подходит и 0.

Наконец, команда document.write() используется для вывода выбранного числа на странице.

Теперь посмотрим на вторую часть сценария:

```
if (num == 0)
   {cliche=var0}
  if (num == 1)
   {cliche=var1}
```

```
if (num == 2)
  {cliche=var2}
document.write(cliche + "<br>") >
```

Условия после IF требуют двойного знака равенства ==.

Если условие верно, будет выполнена команда, заключенная в {фигурные скобки}. Возможны только три результата, поэтому мы написали три условия, чтобы одно из них оказалось верным.

Обратите внимание еще, что условие заключено в (круглые скобки), а результат — в {фигурные}.

Команда document.write(cliche) выведет на странице то изречение, которое было присвоено переменной cliche.

Задание

Измените программу так, чтобы она показывала рисунок, выбранный случайным образом из трех: pic1.gif, pic2.gif и pic3.gif.

2.23 Введение в циклы for

Концепция

Во всех языках программирования имеются средства организации ветвления. В JavaScript для этого используется оператор IF. Во всех языках программирования имеются также средства организации повторяющихся операций или циклов. В JavaScript бывают циклы двух видов: While и For.

Обычно циклы For используются, когда известно количество повторений, а циклы While — когда точно не известно, сколько раз нужно повторить цикл. В данном примере будет рассмотрен цикл For.

Сценарий

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<H3>Посчитаем от одного до пяти:</H3>
<SCRIPT type="text/javascript">
for (i=1; i<=5; i=i+1)
{document.write(i + "<BR>");}
</SCRIPT>
</BODY>
</HTML>
```

Результат работы сценария

На странице столбиком выводятся числа от 1 до 5.

1

2

3

4

5

Разбор сценария

Посмотрим на синтаксис оператора for, for (i=1; i<=5; i=i+1). В нем три части. Все они разделены точкой с запятой. Рассмотрим их по порядку.

i=1 задает начальное значение переменной, управляющей циклом. В данном случае это 1, но можно было бы задать 10 или 100. Это просто начальная точка отсчета цикла.

i<=5 — условие, определяющее, сколько в цикле будет повторений. В нашем случае цикл будет повторяться до тех пор, пока i не станет больше пяти. Мы начали с одного и досчитаем до пяти.

i=i+1 определяет значение шага цикла. В нашем случае программа будет прибавлять 1 к i в конце цикла. Программа может прибавлять и 2, и 20, и 567. Нам просто надо увеличивать i на 1.

Наконец оператор document.write, выводит число на страницу. Обратите внимание на
 в операторе document.write — это заставляет каждое число выводиться в новой строке. С таким же успехом можно было вывести их в одну строку, разделив запятыми и просто изменяя часть текста, которая появляется после каждого числа.

Этот код JavaScript повторится пять раз и выведет на странице цифры от 1 до 5. Мы могли бы заставить его досчитать до миллиона, но это потребовало бы слишком много пространства Web-страницы.

Задание

Напишите документ HTML с текстом "Сюрприз!", заключенным в теги <H2> и расположенным на самом верху страницы.. Начните с белого фона. Потом с помощью JavaScript досчитайте до 50000.

В этот момент цвет фона меняется на желтый и появляется текст: "Скоро будет еще один цветной сюрприз..."

Снова досчитайте до 50 тысяч, и тогда фон должен опять поменяться.

2.24 Введение в циклы while

Концепция

В этом примере рассматривается цикл While. Помните, мы говорили, что циклы For используются, когда известно, сколько раз нужно их повторять, а циклы While — когда не известно. Он покажет, как пользоваться переменными, чтобы сосчитать количество повторений цикла и подготовиться к заданию.

Сценарий

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
 loops=3
 num=1
 while (num <= loops)
   {    document.write("Добро ")
   num=num+1    }
   document.write("Пожаловать!")
  </SCRIPT>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится фраза:

Добро Добро Добро Пожаловать

Разбор сценария

```
<SCRIPT type="text/javascript">
        loops=3
        num=1
       while (num <= loops)
        {             document.write("Добро ")
          num=num+1          }
       document.write("Пожаловать!")
    </SCRIPT>
```

Синтаксис операторов While и For похож. Разница в том, что мы задаем начальное значение индекса и шаг изменения прямо в команде For. Команда While содержит только условие.

while(num<=loops) говорит программе, что она должна повторять цикл, пока значение num меньше или равно значению переменной loops. Видите знак <=? Другими словами, программа повторит цикл трижды, один раз для num=1, один для num=2 и еще один для num=3.

Каждый раз, когда программа выполняет цикл, она пишет "Добро" и прибавляет 1 к num. Когда num доходит до 4, цикл заканчивается. 4 больше 3, поэтому программа напишет "Добро" три раза.

Завершающий штрих — команда document.write со словом "Пожаловать!".

Задание

Измените описанную выше программу так, чтобы пользователь получал запрос: "Сколько раз пожелать Вам Добро пожаловать?" Для ответа создайте переменную. Используйте команду eval(), которая преобразует текст в число? Затем программа должна написать "Добро" столько раз, сколько ее просили.

2.25 Массивы

Концепция

Рассмотрим массивы (array). Каждая переменная содержит в данный момент одно значение, но иногда требуется использовать массив (array) или переменную, которая содержит множество значений.

Разберем пример, в котором программа предлагает пользователю угадать телевизионный канал из перечня телеканалов. Запрос повторяется до тех пор, пока пользователь не угадает. Каждый раз при нажатии кнопки случайным образом выбирается новый телеканал.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/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 телеканал!"
    onClick="whichtv()">
</FORM>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится кнопка с надписью "Угадайте телеканал!", при нажатии на которую выводится окно с предложением угадать любимый канал из перечисленных.

В зависимости от ввода пользователю предлагается попробовать еще раз или программа завершается.

Разбор сценария

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

tv=new Array() объявляет tv как массив объектов. С пустыми (скобками) массив может быть какой угодно длины. Можно также указать длину массива, например, tv=new Array(5).

Помните, что массив может иметь множество значений. Можно представить себе массив в виде таблицы:

	tv			
v[0]	v[1]	v[2]	v[3]	v[4]
РТ	ТР	ВЦ	ТВ	В6

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() случайно 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 функция whichtv():

```
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.

Первым делом функция вызывает другую функцию, picktv(). Таким образом, когда бы вы ни нажали на кнопку, будет выбираться новый телеканал.

Обратите внимание на строку while (tv[num] != guess.toUpperCase()). Метод или действие toUpperCase() (в верхний регистр) используется для перевода всего, что было введено, в верхний регистр символов.

Программа повторяет цикл While, пока пользователь не угадает правильный телеканал. Фрагмент с циклом While должен быть уже вполне знакомым.

Обратите внимание на операторы If и Else. Возможны только два результата: либо вы правы, либо ошибаетесь.

Теперь кнопка, которая все это запускает:

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

В JavaScript есть несколько встроенных массивов. Формы можно хранить в массивах. На форму можно ссылаться с помощью document.myform или document.forms[0], если это первая форма. Массивы всегда начинаются с нуля. Вторая форма будет document.forms[1]. Третья — document.forms[2] и так далее...

Изображения также можно хранить во встроенном массиве. Можно ссылаться на pic1.gif как document.pic1.src или как document.images[0].src. Просто продолжайте следовать схеме, указывая номер в [квадратных скобках].

Задание

Напишите программу JavaScript, которая содержит кнопку с надписью: "Щелкните, чтобы попасть на случайный сайт".

Когда пользователь нажимает ее, выполняется функция, которая выберет случайное число и сайт из массива с помощью команды JavaScript top.location.href = url[num]. top (вершина) — это свойство объекта window, оно относится к главному окну браузера location.href, другой объект и свойство, содержит адрес URL.

2.26 Слайд-шоу

Концепция

В этом примере показано слайд-шоу. Пользователь щелкает по ссылке и переходит к следующей картинке.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
  var num=1
  img1 = new Image ()
  img1.src = "pic1.gif"
  img2 = new Image ()
  img2.src = "pic2.gif"
  img3 = new Image ()
  img3.src = "pic3.gif"
function slideshow()
  {   num=num+1
   if (num==4)
    {num=1}
document.animal.src=eval("img"+num+".src")   }
</SCRIPT>
```

```
</HEAD>
<BODY>
<CENTER>
<IMG SRC="pic1.gif" NAME="animal" BORDER=0>
<p>
<A HREF="JavaScript:slideshow()">
  Щелкните, чтобы увидеть следующую картинку</A>
</CENTER>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится изображение pic1.gif и ссылка "Щелкните, чтобы увидеть следующую картинку", при щелчке на которой изображение изменяется на pic2.gif и так далее.

Разбор сценария

Разделим его на две части:

```
<SCRIPT type="text/javascript">
    var num=1
    img1 = new Image ()
    img1.src = "pic1.gif"
    img2 = new Image ()
    img2.src = "pic2.gif"
    img3 = new Image ()
    img3.src = "pic3.gif"
```

Новый момент! num=1 не находится внутри функции. Да и вообще ни одна команда не находится внутри функции. num —

это переменная. Объявляя переменную в начале сценария JavaScript вне функции, мы делаем ее глобальной переменной, то есть доступной для любой следующей далее функции.

img1 = new Image() объявляет новый объект image (изображение). img1.src= источник объекта, определяет, какая картинка хранится в объекте image. Это стандартная схема. Она должна использоваться, когда задается ряд изображений, как в этом сценарии. pic1.gif хранится в img1.src. pic2.gif хранится вimg2.src.

(В круглых скобках) может содержаться ширина и высота каждого рисунка. Это не обязательно, но желательно, так как ускоряет время загрузки.

Объекты image доступны любой следующей далее функции. Так же, как и переменная num, они находятся вне функции. Они просто перечислены здесь, чтобы их могла использовать любая часть программы. Помещая все объекты изображений вне функции, программа загружает изображения. Это будет еще более важно в следующем примере анимации. Во время анимации пользователь не захочет ждать загрузки каждой картинки с сервера. Поэтому требуется предварительная загрузка.

Теперь вторая часть:

```
function slideshow()
{num=num+1
if (num==4)
  {num=1}
document.animal.src=eval("img"+num+".src")}
</script>
```

</head>

<body>

Начальное значение num равно 1. Это было задано в первом фрагменте. Когда пользователь щелкает на тексте ссылки, запускается функция slideshow.

slideshow() прибавляет к num единицу. Когда num доходит до 4, то num снова присваивается значение 1. document.animal.src меняется на img плюс значение num и плюс .src. Например, если num = 1, то document.animal.src становится img1.src.

Обратите внимание, что команда eval() преобразует "img1.src" в указание на источник изображения. Без нее это был бы простой набор букв.

И, наконец, команда, которая все это запускает в работу :

Щелкните, чтобы увидеть следующую картинку

Здесь вызывается не функция, а функция с префиксом JavaScript? Это позволяет использовать все части сценария, а не только функцию. Если написать здесь только функцию, то рисунков не будет, потому что они останутся за скобками.

Задание

Перепишите приведенный выше сценарий JavaScript. Покажите первым pic1.gif, как в примере. Однако дальше новый сценарий должен показать img3.src (num=3), потом img2.src, потом img1.src. Когда num=0, измените num на 3.

2.27 Анимация

Концепция

Здесь с помощью JavaScript создается анимационная картинка. Для анимации особенно важно, чтобы все изображения были заранее загружены в память компьютера. С помощью команд new Image() и img.src, стоящих вне функции.

Если не позаботиться об этом заранее, то пользователю придется ждать, пока каждая картинка будет загружаться с сервера. Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
 var num=1
 img1 = new Image (150,150)
 img1.src = "pic1.gif"
 img2 = new Image (150,150)
 img2.src = "pic2.gif"
 img3 = new Image (150,150)
 img3.src = "pic3.gif"
  function startshow()
  {   for (i=0; i<21; i=i+1)
{document.mypic.src=eval("img"+num+".src")
    for(x=1; x<8000; x=x+1)
  {}
    num=num+1
    if(num==4)
    {num=1}   }
   document.mypic.src=img1.src   }
```

```
</SCRIPT>
</HEAD>
<BODY>
<CENTER>
<IMG SRC="pic1.gif" NAME="mypic" BORDER=0 alt="">
<p>
<A HREF="JavaScript:startshow()">
  Показать анимацию</a>
</CENTER>
</BODY>
</HTML>
```

Результат работы сценария

На странице выводится изображение pic1.gif и текстовая ссылка " Показать анимацию", при нажатии на которую начинается последовательная смена изображений (анимация).

(Сценарий может не работать в браузере MSIE.)

Разбор сценария

Мы начинаем с предварительной загрузки изображений. Обратите внимание, что это происходит вне функции, так же, как и задание переменной num. Это выглядит следующим образом:

```
<SCRIPT type="text/javascript">
   var num=1
   img1 = new Image (150,150)
   img1.src = "pic1.gif"
   img2 = new Image (150,150)
   img2.src = "pic2.gif"
```

```
    img3 = new Image (150,150)
    img3.src = "pic3.gif"
```

Теперь функция:

```
function startshow()
 {  for (i=0; i<21; i=i+1)
{document.mypic.src=eval("img"+num+".src")
    for(x=1; x<8000; x=x+1)
    {}
    num=num+1
    if(num==4)
     {num=1}    }
  document.mypic.src=img1.src    }
</SCRIPT>
```

startshow() содержит новый элемент, вложенные циклы! Видите, внутри первого цикла for находится второй (слово for повторяется дважды.) Второй цикл замедляет смену картинок, чтобы пользователь смог их разглядеть.

Обратите внимание, в {фигурных скобках} цикла ничего нет. Вложенный цикл считает от 1 до 8000 после появления каждой картинки. На это уходят доли секунды.

Внешний цикл заставляет программу повторяться в цикле 21 раз. Видите это в функции: в первом операторе цикла for (i=0; i<21; i=i+1). Так как анимация состоит из трех картинок, она будет повторена 7 раз: 7 X 3 = 21.

Когда цикл закончится, картинка снова вернется к pic1.gif.

Вот команда, которая помещает первое изображение на страницу:

<IMG SRC="pic1.gif" NAME="mypic" BORDER=0 alt="">

Обратите внимание, что имя задано как "mypic". Если посмотреть на иерархию операторов в сценарии, то оно находится между document и src.

Сссылка, которая запускает анимацию:

<A HREF="JavaScript:startshow()">Показать анимацию

Снова команда HREF указывает на JavaScript:функция(), чтобы все глобальные переменные были доступны.

Задание

Перепишите программу анимации. Пусть первым будет pic1.gif, как в данном примере, но вставьте его в форму. Включите в нее текстовое поле, куда пользователь мог бы ввести слово slow (медленно), medium (умеренно) или fast (быстро), выбирая скорость смены картинок. Пусть medium стоит по умолчанию. 800 будет "быстро", 1600 — "умеренно", 2400 — "медленно". Запускать анимацию должна текстовая ссылка "Показать анимацию".

2.28 Проверка данных в форме

Концепция

В этом примере JavaScript применяется для проверки данных, которые ввел пользователь. Нужно будет ввести в форму свое имя и номер телефона из 7 или 9 знаков (xxxxxxx или xxx-xx-xx). Необходимо также проверить, что первые 3 символа являются цифрами. Это немного сложнее, чем все остальное, но с этим

необходимо разобраться. Проверка данных часто является важной задачей для программистов.

Этот пример возвращает нас к свойству length (длина) и показывает в действии два новых метода: indexOf(), charAt(). Сам сценарий будет длиннее, чем обычно.

Сценарий

```
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function validfn(fnm)
{fnlen=fnm.length
if (fnlen == 0)
{alert("Вы должны ввести свое имя")
document.dataentry.fn.focus()}}
function validphone(phone)
{len=phone.length
digits="0123456789"
if(len != 7 && len != 9)
{alert("Неверное количество знаков в номере")
document.dataentry.phone.focus()}
for(i=0; i<3; i++)
{if (digits.indexOf(phone.charAt(i))<0)
{alert("Это должны быть цифры")
document.dataentry.phone.focus()
break}}}
</SCRIPT>
```

```
</HEAD>
<BODY>
<FORM NAME="dataentry">
<h2>Подтверждение данных</h2>
Введите свое имя:<br>
  <INPUT TYPE="text" NAME="fn"
    onBlur="validfn(fn.value)">
<SCRIPT LANGUAGE="JavaScript">
  document.dataentry.fn.focus()
</SCRIPT>
Введите номер телефона (xxx-xx-xx):<br>
    <INPUT TYPE="text" NAME="phone" SIZE=10 >
<INPUT TYPE="button"  VALUE="Отправить"
     onClick="validphone(phone.value)">
</BODY>
</HTML>
```

Результат работы сценария

На странице выводятся два поля ввода с просьбой ввести свое имя и номер телефона. В случае недопустимого ввода выдается сообщение об ошибке.

Разбор сценария

Сценарий достаточно простой. Имеется две функции, validfn() и validphone(). Одна проверяет, введено ли имя, другая проверяет телефонный номер. Займемся первой:

```
function validfn(fnm)
{  fnlen=fnm.length
```

```
 if (fnlen == 0)
 {alert("Вы должны ввести свое имя")
 document.dataentry.fn.focus()}  }
<body>
<INPUT TYPE="text" NAME="fn"
  onBlur="validfn(fn.value)">
```

Функция validfn(fnm) вызывается обработчиком события onBlur. onBlur запускается, когда курсор переходит на следующий элемент, в данном случае, когда мы выходим из текстового поля fn. Заметьте, что параметр fn.value передается из формы в функцию, где получает новое имя fnm. fn.value можно было бы обозначить как document.dataentry.fn.value, но раз мы находимся в документе и внутри формы, это не обязательно.

Помните, содержимое поля формы передает команда имя_формы.value. Одного имени мало.

Переменной с именем fnlen присвоено значение длины имени пользователя. Таким образом, если пользователь введет имя "Коля", значение fnlen будет равно 4.

Если пользователь не вписал свое имя, значит, длина равна 0. Тогда программа выводит окно с сообщением об ошибке и ставит курсор или фокус в поле для ввода имени. Форма не столько проверяет, правильно ли вписано имя, сколько было ли что-нибудь вписано вообще.

Теперь посмотрим, как программа проверяет телефонный номер:

```
function validphone(phone)
```

```
{len=phone.length
digits="0123456789"
if(len != 7 && len != 9)
{alert("Неверное количество знаков в номере")
document.dataentry.phone.focus()}
for(i=0; i<3; i++)
{if (digits.indexOf(phone.charAt(i))<0)
{alert("Это должны быть цифры")
document.dataentry.phone.focus()
break}}
```

Разберем эту функцию шаг за шагом. Во-первых, длина телефонного номера присваивается переменной len. Переменная digits содержит все десятичные цифры.

Потом оператор If проверяет, равна ли длина номера 7 или 9 знакам, хотя и звучит это несколько неуклюже. Двойной знак && в Javascript означает "проверить оба свойства".

Если условие не выполнено, программа говорит пользователю о том, что он ввел неверное количество цифр, и снова устанавливает курсор или фокус в поле для ввода номера.

for(i=0; i<3; i++) проверяет первые 3 цифры номера одну за другой.

Выражение if (digits.indexOf(phone.charAt(i))<0) знакомит нас с двумя новыми методами: indexOf() и charAt(). Посмотрим на phone.charAt(i). Предположим, телефонный номер 123, и i = 2. Знак на второй позиции — цифра 3.! Порядковые номера начинаем

считать с нуля. Таким образом, phone.charAt(0) = 1, phone.charAt(1) = 2, а phone.charAt(2) = 3.

indexOf —это метод, дающий порядковый номер для заданного значения. С помощью if (digits.indexOf (phone.charAt(i))<0), JavaScript ищет значение phone.charAt(i) в переменной digits.

Если телефонный номер 1234567 и i = 1, то программа ищет вторую цифру в переменной digits и находит ее, возвращая значение 1, так как digits = "0123456789".

Если номер телефона 12д и i = 2, программа ищет "д" в переменной digits. Не найдя ее, она возвращает -1. Если значение = -1 (<0), тогда появляется окно с сообщением об ошибке и курсор или фокус устанавливается на прежнее место. Для телефонного номера xxxxxxx так можно проверить все 7 цифр.

И последнее — код HTML для формы:

```
Введите свое имя:<br>
  <INPUT TYPE="text" NAME="fn"
    onBlur="validfn(fn.value)">
<SCRIPT LANGUAGE="JavaScript">
  document.dataentry.fn.focus()
</SCRIPT>
Введите номер телефона (xxx-xx-xx):<br>
    <INPUT TYPE="text" NAME="phone" SIZE=10>
<INPUT TYPE="button"  VALUE="Отправить"
     onClick="validphone(phone.value)">
```

При использовании JavaScript с формами давайте каждому элементу уникальное имя, которое будет связывать его с разделом JavaScript, который его обрабатывает. Такое связывание мы уже использовали ранее. Просмотрите элементы формы и затем код JavaScript и определите, где одно связано с другим.

Задание

Изучите сегодняшнюю программу и заставьте ее работать. Потом внесите несколько изменений. Попросите ввести телефонный номер в формате xxx-xxxx. Пусть функция validphone(phone) проверит, стоит ли дефис на позиции 3. Команда != в JavaScript означает "не равно". Это может понадобиться.

Литература

1. Бен Хеник, HTML и CSS. Путь к совершенству. СПб: Питер, 2011 -336 с.
2. Дунаев В., HTML, скрипты и стили. СПб: БХВ-Петербург, 2011- 816 с.
3. Дронов В., HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. СПб: БХВ-Петербург, 2011 - 416 с.
4. Джон Поллок. JavaScript. Руководство разработчика. СПб: Питер,2011-544 с.
5. Дэвид Макфарланд. JavaScript. Подробное руководство. Эксмо, 2009-608 с.
6. Чак Муссиано, Билл Кеннеди. HTML и XHTML. Подробное руководство. Символ-Плюс, 2011 - 752 с.
7. Кисленко Н.П. HTML. Самое необходимое. СПб: БХВ-Петербург, 2012 – 352 с.
8. Комолова Н., Яковлева Е. HTML, XHTML и CSS. СПб: Питер, 2012 –304 с.
9. Пол Вилтон, Джереми МакПик. JavaScript. Руководство программиста. СПб: Питер, 2009-720 с.
10. Роберт Агулар. HTML и CSS. Основа любого сайта. Экспо, 2010 –320 с.
11. Стоян Стефанов. JavaScript. Шаблоны. СПб: Символ-плюс, 2011-272с.
12. Климов А. JavaScript на примерах. СПб: БХВ-Петербург, 2009-336 с.

13. Шафер С., HTML, XHTMLи CSS. Библия пользователя. М.: Вильямс, 2010 – 656 с.
14. Лабберс К., Олберс Н., Салим К.. HTML5 для профессионалов:мощные инструменты для разработки современных веб-приложений. М.: Вильямс, 2011 – 272 с.

Интернет-ресурсы

1. www.w3.org
2. www.intuit.ru/studies/courses. - Открытые интернет-курсы «Интуит» по курсу «Информатика»
3. lib.kchgu.ru. - Электронная библиотека КЧГУ | ФГБОУ ВО КЧГУ имени У.Д. Алиева
4. http://javascript.itsoft.ru -Online-справочник по языку JavaScript
5. http://doks.gorodok.net/ - Справочники и учебники по PHP, C/C++, HTML, CSS, Javascript, XML, SQL, Perl, Windows, Unix
6. http://computerlibrary.info/ - Рубрики: Веб-мастеру, Интернет, Софт
7. http://www.webclub.ru/ - WebClub - Всероссийский Клуб Веб-разработчиков
8. http://dweb.ru/rass/html/index.htm - Учебник по HTML
9. http://www.htmlbook.ru/ - посвящен языку HTML, CSS, веб-дизайну, графике и процессу создания сайтов.
10. http://www.codenet.ru - все для программиста
11. http://html.krsk.ru/css/index.asp - Каскадные таблицы стилей

Приложение

1. Стандартные объекты JavaScript

Array Массив пронумерованных элементов, также может служить стеком или очередью

Boolean Объект для булевых значений

Date Функции для работы с датой и временем

Error объект для представления ошибок

EvalError Ошибка при выполнении функции eval

Function Каждая функция в яваскрипт является объектом класса Function.

Math Встроенный объект, предоставляющий константы и методы для математических вычислений.

Number Объект для работы с числами

Object Базовый объект javascript

RangeError Ошибка, когда число не лежит в нужном диапазоне

ReferenceError Ошибку при ссылке на несуществующую переменную

RegExp Позволяет работать с регулярными выражениями.

String Базовый объект для строк. Позволяет управлять текстовыми строками, форматировать их и выполнять поиск подстрок.

SyntaxError Ошибка при интерпретации синтаксически неверного кода

TypeError Ошибка в типе значения

URIError Ошибка при некорректном URI

Объекты браузера

Window Два в одном: глобальный объект и окно браузера

2. Глобальные методы

Alert Выводит модальное окно с сообщением

clearInterval Останавливает выполнение кода, заданное **setIntervalclearTimeout** Отменяет выполнение кода, заданное **setTimeoutconfirm** Выводит сообщение в окне с двумя кнопками: "ОК" и "ОТМЕНА" и возвращает выбор посетителя

decodeURIP аскодирует URI, закодированный при помощи encodeURI

decodeURIComponentРаскодирует URI, закодированный при помощи encodeURIComponent

encodeURIКодирует URI, заменяя каждое вхождение определенных символов на escape-последовательности, представляющие символ в кодировке UTF-8.

encodeURIComponent Кодирует компоненту URI, заменяя определенные символы на соответствующие UTF-8 escape-последовательности

eval Выполняет строку javascript-кода без привязки к конкретному объекту.

isFinite возвращает, является ли аргумент конечным числомisNaN Проверяет, является ли аргумент NaN

parseFloatпре образует строковой аргумент в число с плавающей точкой

parseIntпре образует строковой аргумент в целое число нужной системы счисления

prompt Выводит окно с указанным текстом и полем для пользовательского ввода.

setInterval Выполняет код или функцию через указанный интервал времени

setTimeout Выполняет код или функцию после указанной задержки

3. Синтаксические конструкции

Break Завершает текущий цикл или конструкции switch и label и передает управление на следующий вызов

Continue Прекращает текущую итерацию цикла и продолжает выполнение со следующей итерации

do..while Задает цикл с проверкой условия после каждой итерации

for Создать цикл, указав начальное состояние, условие и операцию обновления состояния

for..in Перебрать свойства объекта, для каждого свойства выполнить заданный код

function Объявить функцию

if Выполняет тот или иной блок кода в зависимости от того, верно ли условие

label Указать идентификатор для использования в break и continue

return Возвратить результат работы функции

switch Сравнивает значение выражения по различными вариантами и при совпадении выполняет соответствующий код

throw Инициировать("бросить") исключение

try..catch Ловить все исключения, выпадающие из блока кода

var Объявить переменную (или несколько) в текущей области видимости

while Задает цикл, который выполняется до тех пор, пока условие верно. Условие проверяется перед каждой итерацией.

With Добавить новую область видимости

Блок Группировка javascript-вызовов внутри фигурных скобок

Тестирование JavaScript

1. Выберите утверждение, характеризующее язык JavaScript:
   - это язык разработки сетевых баз данных
   - это язык описания взаимодействий клиента и сервера
   - это язык управления сценариями просмотра гипертекстовых Web-страниц
2. Язык JavaScript — ...
   - регистро-независимый
   - регистро-зависимый
3. С каких символов может начинаться однострочный комментарий в JavaScript?
   - /*
   - //
   - {
4. Где в HTML-странице можно размещать JavaScript-код?
   - в атрибуте HREF гипертекстовой ссылки (схема URL "JavaScript:")
   - во внешнем файле, подключаемом с помощью <A HREF="имя_файла">ссылка</A>
   - во внешнем файле, подключаемом с помощью <SCRIPT SRC="имя_файла"></SCRIPT>
   - в атрибутах, соответствующих обработчикам событий (например, onClick)
   - между тэгами <SCRIPT> и </SCRIPT>
5. Какой оператор выведет на экран окно предупреждения с текстом Привет?
   - document.print('Привет');
   - document.write('Привет')
   - alert('Привет');
6. Выберите допустимые способы указания языка скрипта:

- <SCRIPT LANGUAGE="JavaScript">
- <SCRIPT LANGUAGE="javascript">
- <SCRIPT LANGUAGE="text/vbscript">
- <SCRIPT TYPE="text/javascript">
- <SCRIPT TYPE="JavaScript">

7. Какие комбинации символов ограничивают многострочный комментарий в JavaScript?

- между { и }
- между <!-- и -->
- между /* и */
- между // и //

8. Интерпретатор какого языка будет использован браузером для выполнения следующего скрипта: <SCRIPT LANGUAGE="JavaScript" TYPE="text/vbscript"> ... </SCRIPT> если браузер "понимает" оба атрибута TYPE и LANGUAGE

- JavaScript
- VBScript√
- использовать одновременно оба атрибута TYPE и LANGUAGE недопустимо

9. Как связаны языки JavaScript и JScript?

- JScript является подмножеством JavaScript
- JavaScript является подмножеством JScript
- это два названия одного и того же языка

10. Выберите верное утверждение: JavaScript является языком ...

- компилируемым
- интерпретируемым

11. Какими символами заканчивается однострочный комментарий?

- //
- }
- конец строки

12. Можно ли в JavaScript строковой переменной присвоить число в качестве значения?

- нельзя, будет выдана ошибка типов
- можно, переменная "на лету" сменит тип и будет хранить число
- можно, при этом число неявно преобразуется в строку

13. Что делает строка massiv = new Array()?

- сравнивается значение переменной massiv с результатом вызова функции Array
- объявляет переменную с именем massiv типа "массив"
- объявляет переменную с именем Array и присваивает ее значение переменной massiv

14. Что будет создано в результате следующего объявления: a = new Array(5);

- массив из одного элемента — числа 5
- массив из 5 элементов, значения которых не определены
- массив из 6 элементов, значения которых не определены

15. Каким будет массив: var a = [35,21,13,8,5,3,2,1] после операции a.sort()

- ["1","2","3","5","8","13","21","35"]
- ["1","13","2","21","3","35","5","8"]
- [1,2,3,5,8,13,21,35]
- [1,13,2,21,3,35,5,8]

16. Что выдаст данный скрипт: var e="2.71828182845", a = e.split('8'); alert(a[2]+8);

- 9
- 10
- 18
- 28

17. Каков результат работы данного скрипта: <SCRIPT> mas=new Array(2); mas3=5; alert(mas[3]); </SCRIPT>

- будет выдано сообщение о синтаксической ошибке
- будет показано окно предупреждения с цифрой 5
- будет показано окно предупреждения с надписью undefined

18. Что называется циклом?

- множество данных, размещенных в одной переменной
- средство организации повторяющихся операций
- ошибка, при которой программа повторяется бесконечное количество раз

19. В каком порядке перечисляются три выражения в описании оператора for

- инициализация_переменных_цикла; модификация_переменных_цикла; условие_продолжения_цикла
- инициализация_переменных_цикла; условие_продолжения_цикла; модификация_переменных_цикла
- условие_продолжения_цикла; инициализация_переменных_цикла; модификация_переменных_цикла

20. Не выполняя скрипт в браузере, определите, что будет выведено в следующем цикле? for(i=0;i<9;i++) { if(i>5) break; document.write(i); }

- ничего
- 01234
- 012345
- 5678
- 678

21. Не выполняя скрипт в браузере, определите, что будет выведено в следующем цикле? for(i=0;i<9;i++) { if(i>5) continue; document.write(i); }

- ничего
- 01234
- 012345
- 5678

- 678

22. Можно ли в JavaScript целочисленной переменной присвоить строку в качестве значения?

- нельзя, произойдет ошибка типов
- можно, только если в этой строке записано число
- можно всегда, переменная "на лету" сменит тип

23. Каким из перечисленных способов нельзя объявить новый массив?

- mas = new Array();
- mas = new Array(5);
- mas[0]=1; mas[1]=2; mas[2]=3;
- mas = new Array('Help');

24. Что выдаст данный скрипт: var a = [35,21,13,8,5,3,2,1]; a.sort(); alert(a[3]+7); ?

- 9
- 10
- 12
- 28
- 217

25. Что выдаст данный скрипт:

```
var p="3.1415926535",  a = p.split('5');
alert(a[2]+5);
```

- 8
- 35
- 931
- 9265

26. Что будет создано в результате следующего объявления: a = new Array(3,'Moscow','London','Paris')

- массив из трех элементов — строк "Moscow", "London" и "Paris"
- массив из четырех элементов — числа 3 и строк "Moscow", "London" и "Paris"

- будет выдана ошибка, т.к. нельзя в массиве смешивать элементы разных типов

27. Выберите верные утверждения:

- тело цикла for обязательно выполняется хотя бы раз
- тело цикла for может не выполниться ни разу
- тело цикла while обязательно выполняется хотя бы раз
- тело цикла while может не выполниться ни разу

28. Не выполняя скрипт в браузере, определите, что будет выведено в следующем цикле? for(i=0;i<9;i++) { document.write(i); if(i>5) break; }

- 01234
- 012345
- 0123456
- 5678
- 678

29. Не выполняя скрипт в браузере, определите, что будет выведено в следующем цикле? i=0; while(i<9) { if(i>3) break; document.write(i); i++; }

- ничего
- 01234
- 0123
- 5678
- 678

30. Не выполняя скрипт в браузере, определите, что будет выведено в следующем цикле? for(i=0;i<9;i++) { document.write(i); if(i<5) break; }

- 01234
- 012345
- 0
- 5678

31. Каким образом внутри некоторой функции g() получить доступ к значениям локальных переменных некоторой другой функции f()? (ситуацию "g() описана внутри f()" не рассматривать)

- для этого нужно объявить в функции g() требуемую переменную оператором var
- локальные переменные функции f() доступны в любых функциях по-умолчанию
- это невозможно, локальные переменные функции f() недоступны вне этой функции

32. Укажите допустимые способы обращения к форме с именем anketa:

- document.anketa
- window.anketa
- window.document.anketa
- document.forms['anketa']
- document.forms.anketa

33. В форме с именем fio имеется поле с именем familia, у которого есть свойство value. Каким образом можно обратиться к этому свойству внутри оператора with(document.fio)?

- familia.value
- familia['value']
- fio.familia(value)
- document.fio.familia.value

34. Является ли объект navigator свойством объекта window?

- нет
- да
- да, но только в Netscape Navigator

35. В какой момент создаются объекты в HTML-документе?

- существуют всегда
- в момент загрузки документа
- в результате работы JavaScript-программы

36. Чему в объектной модели документа могут соответствовать атрибуты HTML-контейнера?

- методам объекта

- свойствам объекта
- объектам
- событиям

37. Каков результат работы скрипта:

```
L = window.location;
W = document.write;
W(L);
```

?

- будет выдана ошибка: после write нужны скобки и аргумент
- будет выведено [object], т.к. location — это объект
- будет выведена строка "window.location"
- будет выведен адрес текущей страницы

38. Каков результат работы скрипта: var Z = Math.cos, A = alert, P = Math.PI; A(Z(P)); ?

- будет выдана ошибка: после alert и cos нужны скобки и аргумент
- будет выдана ошибка: в двух местах вместо запятой нужна точка с запятой
- будет выведена буква P в окне предупреждения
- будет выведено число –1 в окне предупреждения
- будет выведено число 1 в окне предупреждения

39. Внутри функции заведена переменная оператором var d. Можно ли использовать переменную d вне этой функции?

- можно, т.к. язык JavaScript — слабо типизирован, ввиду чего в нем все переменные являются глобальными
- можно, но это будет другая переменная с независимым значением

нельзя, т.к. локальные переменные извне не видны

40. Может ли в DOM свойство объекта само быть объектом?

- да
- нет

41. Что из перечисленного является событием?

- клик по гипертекстовой ссылке
- нажатие на кнопку
- завершение загрузки документа
- изменение цвета фона страницы

42. Какой из объектов модели DOM — самый старший?

- navigator
- window
- document

43. Каков результат работы скрипта: var A = alert; A(window.location); ?

- будет выдана ошибка: после alert нужны скобки и аргумент
- будет выведен адрес текущей страницы в окне предупреждения
- будет выведена строка "window.location" в окне предупреждения
- будет выведено [object], т.к. location — это объект

44. Пусть описана глобальная переменная оператором var s. Можно ли изменять ее значение внутри какой-либо функции?

- можно, предварительно объявив ее внутри функции (оператором var s)
- можно, если только внутри этой функции не объявлено (оператором var s) локальной переменной с таким же именем
- нельзя, т.к. внутри функций можно менять лишь значения локальных переменных

45. Укажите НЕдопустимый способ обращения к форме с именем anketa:

- document.anketa
- window.anketa
- window.document.anketa
- document.forms['anketa']
- document.forms.anketa

46. В форме с именем fio имеется поле с именем familia, у которого есть свойство value. Каким образом нельзя обратиться к этому свойству внутри оператора with(document.fio)?

- familia.value
- familia['value']
- familia.value.fio
- document.fio.familia.value

47. Чему в объектной модели документа соответствуют HTML-контейнеры (элементы)?

- методам объекта
- свойствам объекта
- объектам
- событиям

48. Каков результат работы скрипта: var M = Math, A = alert, pi = M.PI/2; A(M.sin(pi)); ?

- будет выдана ошибка: после alert нужны скобки и аргумент
- будет выдана ошибка: в двух местах вместо запятой нужна точка с запятой
- будет выведена буква Q в окне предупреждения
- будет выведено число 0 в окне предупреждения
- будет выведено число 1 в окне предупреждения

49. В каком месте HTML документа может располагаться JavaScript код?

- В секции <head>
- В секции <body>
- В секции <head> и в секции <body>

50. Выберите JavaScript команду позволяющую вывести текст на страницу.

- write('Текст выведен с помощью JavaScript')
- document.write('Текст выведен с помощью JavaScript

- text('Текст выведен с помощью JavaScript')

51 Выберите комментарий использующийся в JavaScript.

- // Я являюсь комментарием
- <!-- Я являюсь комментарием -->
- <? Я являюсь комментарием ?>

52. Чувствителен ли JavaScript к регистру символов?

- Да
- Нет

53. Выберите JavaScript команду создающую строковую переменную.

- var str=new Array("Строковая переменная")
- var str="Строковая переменная"
- var str=new Object("Строковая переменная")

54. Какое событие позволяет выполнять код после щелчка мыши?

- mouseout
- mouseclick
- onmouseclick
- onclick

55. Выберите синтаксически корректную JavaScript команду для вызова функции "callFunction()".

- function callFunction()
- callFunction()
- new callFunction()

56. Выберите перечень содержащий только действительно существующие в JavaScript циклы.

- loop, for, while
- for, while, do..while
- while, for..in, cycle
- circle, while, switch

57. Выберите синтаксически корректную команду для создания объекта JavaScript.

- var obj=create Object
- var obj=call Object
- var obj=new Object()
- var obj=Object()

58. Какой BOM объект содержит информацию о браузере пользователя?

- History
- Browser
- Navigator

59. Выберите синтаксически корректную команду для создания массива.

- var ar=new Array()
- var ar={}
- var ar=new String()
- var ar=create Array()

60. Выберите метод JavaScript позволяющий выполнять произвольный код через заданные промежутки времени.

- callCode()
- timer()
- setInterval()
- setTimeOut()

61. Какое выражение позволяет проверять участки кода на наличие ошибок?

- error..
- catch try..
- catch error..
- try throw

62. Выберите синтаксически корректную условную конструкцию.

- if (a==3) then document.write('Привет')
- if (a==3) document.write('Привет')

- if (a==3) make document.write('Привет')

63. Выберите метод позволяющий округлить число до ближайшего целого.

- ceil
- floor
- round
- random

64. Какой BOM объект хранит информацию о URL текущего документа?

- History
- Browser
- URLinfo
- location

65. Укажите название встроенного JavaScript объекта для работы с датой и временем.

- datetime
- calendar
- date

66. Имеет ли JavaScript встроенную поддержку регулярных выражений?

- Да
- Нет

67. Выберите JavaScript команду для вызова окна оповещения.

- window()
- confirm()
- alert()
- show()

Справочник JavaScript

1. Встроенные объекты JavaScript

Объект	Описание
Объект String	Используется для хранения и обработки текстовой информации.
Объект Array	Позволяет сохранять и обрабатывать группы связанных переменных.
Объект Math	Используется для выполнения математических операций и извлечения значений констант.
Объект Date	Позволяет производить различные операции с датой и временем.
Объект Number	Используется для работы с числами.
Объект Boolean	Используется для работы с логическими значениями (true, false).
Регулярные выражения	Позволяет производить гибкий поиск слов и выражений в тексте.
Глобальные свойства и методы	Свойства и методы данной группы являются глобальными т.е. присутствуют у всех существующих объектов JavaScript.

2. BOM объекты

Объект	Описание
Объект Window	Позволяет узнавать информацию и управлять поведением текущего окна браузера.
Объект Navigator	Позволяет узнать информацию о браузере, который использует пользователь.
Объект Location	Позволяет узнать информацию о текущем URL.
Объект History	Содержит список предыдущих URL, которые были посещены пользователем в данном окне браузера.
Screen	Позволяет узнать информацию о разрешении экрана пользователя.

3. Свойства объекта Document

С помощью данного объекта Вы сможете добавлять, изменять и удалять HTML элементы на странице из скриптов.

Свойство	Описание

anchors	Возвращает массив содержащий все закладки имеющиеся на странице.
cookie	Возвращает cookie связанные с данным документом.
document.doctype	Позволяет узнать doctype документа.
domain	Возвращает доменное имя сервера, на котором размещается данный документ.
forms	Возвращает массив содержащий все формы имеющиеся на странице.
images	Возвращает массив содержащий все картинки имеющиеся на странице.
document.lastModified	Позволяет узнать когда последний раз был модифицирован документ.
links	Возвращает массив содержащий все ссылки имеющиеся на странице.
referrer	Возвращает URL страницы с которой был совершен переход на данную.
document.readyState	Позволяет узнать статус загрузки документа.
title	Устанавливает или возвращает заголовок документа.
URL	Возвращает текущий URL.

4. Методы объекта Document

Метод	Описание
createElement	Создает элемент.
getElementById	Позволяет обратиться к элементу с указанным id.
getElementsByName	Позволяет обратиться ко всем элементам на странице с указанным именем.
getElementsByTagName	Позволяет обратиться ко всем указанным тэгам на странице.
write	Выводит переданный текст на страницу.
writeln	Выводит переданный текст на страницу, отступая при этом новую строку, после каждого вывода.

5. События

События - это функции, которые могут быть привязаны к элементам HTML страниц. Код событий выполнится только после того, как произойдет их активирующее действие. Разные типы событий имеют разные активирующие действия. DOM объект **events** содержит дополнительную информацию о событиях, которые произошли.

Виды событий

Название	Описание
onblur	Код переданный данному событию исполнится после того, как элемент перестанет быть активным.
onchange	Код переданный данному событию исполнится после того, как содержимое данного элемента будет изменено.
onclick	Код переданный данному событию исполнится после того, как на данном элементе будет произведен щелчок мыши.
ondblclick	Код переданный данному событию исполнится после того, как на данном элементе будет произведен двойной щелчок мыши.
onerror	Код переданный данному событию исполнится если при загрузки документа или картинки произойдет ошибка.
onfocus	Код переданный данному событию исполнится после того, как элемент станет активным.
onkeypress	Код переданный данному событию исполнится после того, как будет произведено нажатие на клавишу клавиатуры.
onkeyup	Код переданный данному событию исполнится после того, как нажатая клавиша будет отпущена.
onload	Код переданный данному событию исполнится после того, как картинка или страница полностью загрузится.
onmousedown	Код переданный данному событию исполнится после того, как будет нажата клавиша мыши.
onmouseout	Код переданный данному событию исполнится после того, как курсор мыши будет выведен за пределы элемента.
onmouseover	Код переданный данному событию исполнится после того, как курсор мыши будет наведен на элемент.
onmouseup	Код переданный данному событию исполнится после того, как будет отпущена нажатая кнопка мыши.
onselect	Код переданный данному событию исполнится после того,

	как текст элемента будет выделен.
onunload	Код переданный данному событию исполнится после того, как страница будет закрыта.

Атрибуты событий

Атрибуты событий хранятся в DOM объекте **events**. С помощью атрибутов событий Вы можете узнать дополнительную информацию о вызове события. Подробнее о том как можно обратится к атрибутам событий из скриптов рассказано ниже:После того, как произошло событие создается DOM объект event содержащий атрибуты событий.

1. Объект event должен быть передан функции, которая будет выполнятся. Пример:**onclick='funcname(event).**'.
2. При объявлении функции необходимо указать, что она принимает объект event. Пример: **function funcname(event){Код функции}.**
3. Теперь Вы можете обращаться к атрибутам событий в коде функции. Пример: **alert(event.shiftKey)**.

Таблица атрибутов событий:

Атрибут	Описание
altKey	Позволяет узнать была ли нажата клавиша Alt во время вызова события.
button	Позволяет узнать какая клавиша мыши была нажата во время вызова события. Атрибут имеет значение 0 если была нажата левая кнопка мыши, 1 если была нажата средняя клавиша мыши и 2 если была нажата правая кнопка мыши.
clientX	Позволяет узнать горизонтальные координаты указателя мыши во время вызова события относительно границ документа.
clientY	Позволяет узнать вертикальные координаты указателя мыши во время вызова события относительно границ документа.
ctrlKey	Позволяет узнать были ли нажата клавиша Ctrl во время вызова события.
screenX	Позволяет узнать горизонтальные координаты указателя мыши во время вызова события относительно границ экрана.
screenY	Позволяет узнать вертикальные координаты указателя мыши во время вызова события относительно границ экрана.

shiftKey	Позволяет узнать были ли нажата клавиша Shift во время вызова события.
target	Позволяет узнать элемент который вызвал событие.
type	Позволяет узнать имя события.

6. Стандартные свойства и методы

В DOM для каждого HTML элемента на странице создается соответствующий объект. Ниже перечислены стандартные свойства и методы, которые присутствуют у всех объектов в DOM. Помимо стандартных свойств и методов существуют еще и специальные свойства методы, которые присутствуют только у определенных объектов: список специальных свойств и методов

Стандартные свойства

Свойство	Описание
attributes	Возвращает массив содержащий все атрибуты элемента.
childNodes	Возвращает массив содержащий все узлы потомки элемента.
className	Позволяет установить или узнать значение атрибута class элемента.
dir	Позволяет установить или узнать направление текста элемента.
id	Позволяет установить или узнать значение атрибута id элемента.
innerHTML	Позволяет установить или узнать текстовое содержимое (позволяет обратится к свойству текстового узла) элемента.
lastChild	Позволяет обратится к последнему потомку элемента.
nextSibling	Позволяет обратится к следующему элементу на данном уровне иерархии (позволяет обратится к следующему узлу брату).
nodeName	Возвращает имя узла.
nodeType	Возвращает тип узла.
nodeValue	Возвращает значение узла.
parentNode	Позволяет обратится к родительскому узлу элемента.

style	Позволяет установить или узнать значения свойств стиля элемента.

Стандартные методы

Метод	Описание
appendChild	Добавляет узел потомок к элементу.
blur	Делает элемент неактивным.
click	Производит щелчок по данному элементу.
focus	Делает элемент активным.
getAttribute	Возвращает значение указанного атрибута.
hasChildNodes	Позволяет узнать имеет ли элемент узлы потомки.
removeAttribute	Удаляет указанный атрибут.
removeChild	Удаляет указанный узел потомок.
setAttribute	Позволяет добавить новый атрибут.

7. Специальные методы и свойства DOM объектов

Существуют некоторые HTML элементы для которых в объектной структуре создаются объекты (узлы), которые помимо набора стандартных DOM свойств и методов имеют еще и специальные DOM свойства и методы присущие только данному объекту.

Таблица специальных DOM методов и свойств разных объектов

В таблице ниже перечислены специальные DOM свойства и методы различных DOM объектов.

Элемент	Специальные DOM методы и свойства	Описание
<a>	**charset**	Возвращает или устанавливает значение атрибута charset ссылки.
	href	Возвращает или устанавливает значение атрибута href ссылки.
	name	Возвращает или устанавливает значение атрибута name ссылки.
	target	Возвращает или устанавливает значение атрибута target ссылки.

	type	Возвращает или устанавливает значение атрибута type ссылки.
<area>	**alt**	Возвращает или устанавливает значение атрибута alt данного элемента.
	coords	Возвращает или устанавливает значение атрибута coords данного элемента.
	hash	Возвращает или устанавливает значение атрибута hash данного элемента.
	host	Возвращает или устанавливает значение атрибута host данного элемента.
	hostname	Возвращает или устанавливает значение атрибута hostname данного элемента.
	href	Возвращает или устанавливает значение атрибута href данного элемента.
	noHref	Возвращает или устанавливает значение атрибута nohref данного элемента.
	pathname	Возвращает или устанавливает значение атрибута pathname данного элемента.
	port	Возвращает или устанавливает значение атрибута port данного элемента.
	protocol	Возвращает или устанавливает значение атрибута protocol данного элемента.
	search	Возвращает или устанавливает значение атрибута search данного элемента.
	shape	Возвращает или устанавливает значение атрибута shape данного элемента.
	target	Возвращает или устанавливает значение атрибута target данного элемента.
<button>	**form**	Позволяет обратится к форме частью которой является данная кнопка.
	name	Возвращает или устанавливает значение атрибута name данного элемента.
	type	Возвращает или устанавливает значение атрибута type данного элемента.
	value	Возвращает или устанавливает значение атрибута value данного элемента.
<form>	**acceptCharset**	Возвращает или устанавливает значение

		атрибута accept-charset данного элемента.
	action	Возвращает или устанавливает значение атрибута action данного элемента.
	enctype	Возвращает или устанавливает значение атрибута enctype данного элемента.
	elements	Возвращает массив который позволяет обратится ко всем элементам данной формы.
	enctype	Возвращает или устанавливает значение атрибута enctype данного элемента.
	lenght	Возвращает количество элементов данной формы.
	method	Возвращает или устанавливает значение атрибута method данного элемента.
	name	Возвращает или устанавливает значение атрибута name данного элемента.
	target	Возвращает или устанавливает значение атрибута target данного элемента.
	reset()	Сбрасывает содержимое всех полей формы. Действие данного метода аналогично нажатию кнопки reset.
	submit()	Отправляет содержимое формы на сервер. Действие данного метода аналогично нажатию кнопки submit.
<iframe>	**align**	Возвращает или устанавливает значение атрибута align данного элемента.
	contentDocument	Возвращает объект document созданный для страницы открытой в данном iframe.
	frameBorder	Возвращает или устанавливает значение атрибута frameborder данного элемента.
	height	Возвращает или устанавливает значение атрибута height данного элемента.
	longDesc	Возвращает или устанавливает значение атрибута longDesc данного элемента.
	marginHeight	Возвращает или устанавливает значение атрибута marginheight данного элемента.

	marginWidth	Возвращает или устанавливает значение атрибута marginwidth данного элемента.
	name	Возвращает или устанавливает значение атрибута name данного элемента.
	scrolling	Возвращает или устанавливает значение атрибута scrolling данного элемента.
	src	Возвращает или устанавливает значение атрибута src данного элемента.
	width	Возвращает или устанавливает значение атрибута width данного элемента.
<input>	**accept**	Возвращает или устанавливает значение атрибута accept (атрибут accept может присутствовать только у элементов с type='file').
	checked	Возвращает или устанавливает значение атрибута checked данного элемента (атрибут checked может присутствовать только у элементов input с type='checkbox' и type='radio').
	defaultChecked	Возвращает true если данный элемент выбран по умолчанию (т.е. если его атрибут checked='checked') и false если нет.
	form	Позволяет обратится к элементам формы частью которой является данный элемент.
	maxLength	Возвращает или устанавливает значение атрибута maxlength данного элемента (атрибут maxlength может присутствовать только у элементов input с type='text' и type='password').
	name	Возвращает или устанавливает значение атрибута name данного элемента.
	readOnly	Возвращает true если содержимое данного поля нельзя изменить (содержимое доступно только для чтения) и false в обратном случае (атрибут readonly запрещающий редактирование поля может быть задан только у элементов input с type='text' и

		type='password'.
	size	Возвращает или устанавливает значение атрибута size данного элемента.
	type	Возвращает или устанавливает значение атрибута type данного элемента.
	value	Возвращает или устанавливает значение атрибута value данного элемента.
	select()	Позволяет выделить текст элемента. Данный метод может быть применен только к элементам input с type='text' и type='password'
<img>	**align**	Возвращает или устанавливает значение атрибута align данного элемента.
	alt	Возвращает или устанавливает значение атрибута alt данного элемента.
	border	Возвращает или устанавливает значение атрибута border данного элемента.
	height	Возвращает или устанавливает значение атрибута height данного элемента.
	hspace	Возвращает или устанавливает значение атрибута hspace данного элемента.
	longDesc	Возвращает или устанавливает значение атрибута longdesc данного элемента.
	name	Возвращает или устанавливает значение атрибута name данного элемента.
	src	Возвращает или устанавливает значение атрибута src данного элемента.
	useMap	Возвращает или устанавливает значение атрибута useMap данного элемента.
	vspace	Возвращает или устанавливает значение атрибута vspace данного элемента.
	width	Возвращает или устанавливает значение атрибута width данного элемента.
<link>	**charset**	Возвращает или устанавливает значение атрибута charset данного элемента.
	href	Возвращает или устанавливает значение атрибута href данного элемента.

	hreflang	Возвращает или устанавливает значение атрибута hreflang данного элемента.
	media	Возвращает или устанавливает значение атрибута media данного элемента.
	rel	Возвращает или устанавливает значение атрибута rel данного элемента.
	rev	Возвращает или устанавливает значение атрибута rev данного элемента.
	target	Возвращает или устанавливает значение атрибута target данного элемента.
	type	Возвращает или устанавливает значение атрибута type данного элемента.
<meta>	**content**	Возвращает или устанавливает значение атрибута content данного элемента.
	httpEquiv	Возвращает или устанавливает значение атрибута httpequiv данного элемента.
	name	Возвращает или устанавливает значение атрибута name данного элемента.
	scheme	Возвращает или устанавливает значение атрибута scheme данного элемента.
<option>	**form**	Позволяет обратится к форме частью которой является данный элемент.
	index	Возвращает позицию данного элемента в выпадающем списке.
	selected	Возвращает или устанавливает значение атрибута selected данного элемента.
	value	Возвращает или устанавливает значение атрибута value данного элемента.
<select>	**form**	Позволяет обратится к форме частью которой является данный элемент.
	length	Возвращает количество элементов в выпадающем списке.
	multiple	Возвращает или устанавливает значение атрибута multiple данного элемента.
	name	Возвращает или устанавливает значение атрибута name данного элемента.
	option	Возвращает массив позволяющий

		обратится ко всем пунктам в выпадающем списке.
	selectedIndex	Возвращает позицию (индекс) выбранного элемента списка.
	add()	Позволяет добавить элемент в выпадающий список.
	remove()	Позволяет удалить элемент из выпадающего списка.
<table>	**border**	Возвращает или устанавливает значение атрибута border данного элемента.
	cellPadding	Возвращает или устанавливает значение атрибута cellpadding данного элемента.
	cells	Возвращает массив позволяющий обратится к любой ячейки в строке.
	cellSpacing	Возвращает или устанавливает значение атрибута cellspacing данного элемента.
	frame	Возвращает или устанавливает значение атрибута frame данного элемента.
	rows	Возвращает массив который позволяет обратится ко всем строкам таблицы.
	rules	Возвращает или устанавливает значение атрибута rules данного элемента.
	summary	Возвращает или устанавливает значение атрибута summary данного элемента.
	width	Возвращает или устанавливает значение атрибута width данного элемента.
	createCaption()	Создает табличный заголовок.
	deleteCaption()	Удаляет табличный заголовок.
	deleteRow()	Позволяет удалить строку из таблицы.
	insertRow()	Позволяет добавить строку в таблицу.
<td>	**abbr**	Возвращает или устанавливает значение атрибута abbr данного элемента.
	cellIndex	Возвращает позицию ячейки в строке.
<tr>	**cells**	Возвращает массив позволяющий обратится ко всем ячейкам строки.
	rowIndex	Позволяет узнать позицию данной

		строки в таблице.
	deleteCell()	Удаляет ячейку из таблицы.
	insertCell()	Вставляет ячейку в таблицу.
<textarea>	**cols**	Возвращает или устанавливает значение атрибута cols данного элемента.
	defaultValue	Возвращает или устанавливает текст по умолчанию.
	disabled	Возвращает или устанавливает значение атрибута disabled данного элемента.
	form	Позволяет обратится к узлу формы частью которой является данный элемент.
	name	Возвращает или устанавливает значение атрибута name данного элемента.
	readOnly	Возвращает или устанавливает значение атрибута readonly данного элемента.
	rows	Возвращает или устанавливает значение атрибута rows данного элемента.
	value	Возвращает или устанавливает текст данного элемента.
	select()	Позволяет выделить текст данного элемента.

8. Свойства объекта Style

Подгруппа свойств	Свойства	Описание
Оформление фона	**background**	Возвращает или устанавливает значение CSS свойства background.
	backgroundAttachment	Возвращает или устанавливает значение CSS свойства background-attachment.
	backgroundColor	Возвращает или

		устанавливает значение CSS свойства background-color.
	backgroundImage	Возвращает или устанавливает значение CSS свойства background-image.
	backgroundPosition	Возвращает или устанавливает значение CSS свойства backround-position.
	backgroundRepeat	Возвращает или устанавливает значение CSS свойства backround-repeat.
Оформление текста	**color**	Возвращает или устанавливает значение CSS свойства color.
	font	Возвращает или устанавливает значение CSS свойства font.
	fontFamily	Возвращает или устанавливает значение CSS свойства font-family.
	fontSize	Возвращает или устанавливает значение CSS свойства font-size.
	fontStyle	Возвращает или устанавливает значение CSS свойства font-style.
	fontVariants	Возвращает или устанавливает

		значение CSS свойства font-variants.
	fontWeight	Возвращает или устанавливает значение CSS свойства font-weight.
	letterSpacing	Возвращает или устанавливает значение CSS свойства letter-spacing.
	lineHeight	Возвращает или устанавливает значение CSS свойства line-height.
	textAlign	Возвращает или устанавливает значение CSS свойства text-align.
	textIndent	Возвращает или устанавливает значение CSS свойства text-indent.
	whiteSpace	Возвращает или устанавливает значение CSS свойства white-space.
	wordSpacing	Возвращает или устанавливает значение CSS свойства word-spacing.
Оформление границ	**border**	Возвращает или устанавливает значение CSS свойства border.
	borderBottom	Возвращает или устанавливает

		значение CSS свойства border-bottom.
	borderBottomColor	Возвращает или устанавливает значение CSS свойства border-bottom-color.
	borderBottomStyle	Возвращает или устанавливает значение CSS свойства border-bottom-style.
	borderBottomWidth	Возвращает или устанавливает значение CSS свойства border-bottom-width.
	borderColor	Возвращает или устанавливает значение CSS свойства border-color.
	borderLeft	Возвращает или устанавливает значение CSS свойства border-left.
	borderLeftColor	Возвращает или устанавливает значение CSS свойства border-left-color.
	borderLeftStyle	Возвращает или устанавливает значение CSS свойства border-left-style.
	borderLeftWidth	Возвращает или устанавливает значение CSS свойства border-left-

		width.
	borderRight	Возвращает или устанавливает значение CSS свойства border-right.
	borderRightColor	Возвращает или устанавливает значение CSS свойства border-right-color.
	borderRightStyle	Возвращает или устанавливает значение CSS свойства border-right-style.
	borderRightWidth	Возвращает или устанавливает значение CSS свойства border-right-width.
	borderStyle	Возвращает или устанавливает значение CSS свойства border-style.
	borderTop	Возвращает или устанавливает значение CSS свойства border-top.
	borderTopColor	Возвращает или устанавливает значение CSS свойства border-top-color.
	borderTopStyle	Возвращает или устанавливает значение CSS свойства border-top-style.
	borderTopWidth	Возвращает или

		устанавливает значение CSS свойства border-top-width.
	borderWidth	Возвращает или устанавливает значение CSS свойства border-width.
Оформление внешнего отступа	**margin**	Возвращает или устанавливает значение CSS свойства margin.
	marginBottom	Возвращает или устанавливает значение CSS свойства margin-bottom.
	marginLeft	Возвращает или устанавливает значение CSS свойства margin-left.
	marginRight	Возвращает или устанавливает значение CSS свойства margin-right.
	marginTop	Возвращает или устанавливает значение CSS свойства margin-top.
Оформление внутреннего отступа	**padding**	Возвращает или устанавливает значение CSS свойства padding.
	paddingBottom	Возвращает или устанавливает значение CSS свойства padding-bottom.
	paddingLeft	Возвращает или

		устанавливает значение CSS свойства padding-left.
	paddingRight	Возвращает или устанавливает значение CSS свойства padding-right.
	paddingTop	Возвращает или устанавливает значение CSS свойства padding-top.
Оформление расположения элементов	**clear**	Возвращает или устанавливает значение CSS свойства clear.
	clip	Возвращает или устанавливает значение CSS свойства clip.
	cssFloat	Возвращает или устанавливает значение CSS свойства float.
	cursor	Возвращает или устанавливает значение CSS свойства cursor.
	direction	Возвращает или устанавливает значение CSS свойства direction.
	display	Возвращает или устанавливает значение CSS свойства display.
	height	Возвращает или устанавливает значение CSS

		свойства height.
	maxHeight	Возвращает или устанавливает значение CSS свойства max-height.
	maxWidth	Возвращает или устанавливает значение CSS свойства max-width.
	overflow	Возвращает или устанавливает значение CSS свойства overflow.
	verticalAlign	Возвращает или устанавливает значение CSS свойства verticalAlign.
	visibility	Возвращает или устанавливает значение CSS свойства visibility.
	width	Возвращает или устанавливает значение CSS свойства width.
Свойства позиционирования	**bottom**	Возвращает или устанавливает значение CSS свойства bottom.
	left	Возвращает или устанавливает значение CSS свойства left.
	position	Возвращает или устанавливает значение CSS свойства position.
	right	Возвращает или устанавливает

		значение CSS свойства right.
	top	Возвращает или устанавливает значение CSS свойства top.
	zIndex	Возвращает или устанавливает значение CSS свойства z-index.
Оформление таблиц	**borderCollapse**	Возвращает или устанавливает значение CSS свойства border-collapse.
	borderSpacing	Возвращает или устанавливает значение CSS свойства border-spacing.
	captionSide	Возвращает или устанавливает значение CSS свойства caption-side
	emptyCell	Возвращает или устанавливает значение CSS свойства empty-cell.
Оформление списков	**listStyle**	Возвращает или устанавливает значение CSS свойства list-style.
	listStyleImage	Возвращает или устанавливает значение CSS свойства list-style-image.
	listStylePosition	Возвращает или устанавливает значение CSS

		свойства list-style-position.
	listStyleType	Возвращает или устанавливает значение CSS свойства list-style-type.
Оформление внешней линии	**outline**	Возвращает или устанавливает значение CSS свойства outline.
	outlineColor	Возвращает или устанавливает значение CSS свойства outline-color.
	outlineStyle	Возвращает или устанавливает значение CSS свойства outline-style.
	outlineWidth	Возвращает или устанавливает значение CSS свойства outline-width.

Содержание

Лепшокова Аланида Нориевна,

Эльканова Айшат Амыровна

Практикум по разработке
web-приложений на языке JavaScript
Учебное пособие

План университета 2016, поз. 12

Редактор Н.В. Ефрюкова

Корректор М.М. Бостанова

Компьютерная вёрстка и набор А.М. Узденова

Подписано в печать 25.10.2016

Формат 60x84/16

Бумага газетная

Объем: 8 усл.печ.л.

Тираж 100 экз.

Издательство Карачаево-Черкесского
государственного университета:
369202, г. Карачаевск, ул. Ленина, 29.
ЛР №040310 от 21.10.1997.

Отпечатано в типографии Карачаево-Черкесского
государственного университета
369202, г. Карачаевск, ул. Ленина, 46.