

# Лабораторная работа № 3. JavaScript.

## 1. Цель работы

Изучить основы программирования на JavaScript.

## 2. Основы JavaScript

При создании документов для WWW-серверов можно воспользоваться различными средствами. Самое распространенное из них - это язык HTML, но возможности этого языка ограничены созданием документов, содержащих статические тексты, таблицы и изображения. Для разработки WEB-страниц, на которых ведется диалог с клиентом, выполняется анимация изображений, реализуется контекстно-зависимый текстовый и графический ввод-вывод, используются другие средства, расширяющие возможности HTML. Среди них - CGI (Common Gateway Interface), Java и JavaScript.

### Типы данных, переменные и литералы

*Типы данных.* JavaScript распознает следующие типы:

- Numbers - числовые значения, такие как 17 или 3.1415928
- Logical (Boolean) - логические: true или false
- Strings - строки, такие как "Hallo !"
- null - специальное ключевое слово для обозначения пустого значения.

В языке JavaScript нет явного различия между целыми и действительными данными. Хотя в языке не определен тип работы с датами, он предоставляет возможность использовать объект date и его методы.

В JavaScript имеются специальные функции для манипулирования строковыми и числовыми значениями:

- - eval - конвертирует, если возможно, строковое значение в тип number;
- - parseInt - преобразует строковое значение в целое, если это возможно;
- - parseFloat - преобразует строковое значение в число с плавающей точкой, если это возможно.

Имена *переменных* в JavaScript должны начинаться с латинской буквы или подчеркивания ('\_'), в качестве последующих символов идентификаторов могут использоваться латинские буквы, цифры (0-9) или знак подчеркивания. Следует учитывать, что JavaScript чувствителен к регистру, то есть имена MyName, MYNAME и myname будут интерпретированы как три различных имени.

Переменные можно объявить двумя способами: просто назначая это значение, например, x = 42 или же с ключевым словом var, например, var x = 19.

Различают *глобальные* и *локальные* переменные. Если переменная объявлена вне функции, то это глобальная переменная, и ее можно использовать везде в текущем документе. Локальная переменная доступна только в пределах функции. Использование ключевого слова var является обязательным только если есть необходимость иметь локальную переменную с таким же именем как у глобальной. Возможен доступ к глобальным переменным одного окна или фрейма в другом окне или фрейме через их имя. Например, если переменная phoneNumber объявлена в родительском фрейме, можно обратиться к этой переменной из дочернего

фрейма как parent.phoneNumber.

*Литералы* - это константы различных типов.

*Целые литералы.* Целочисленные значения могут выражаться десятичным числом (основание системы счисления-10), шестнадцатеричным (16), и восьмеричным (8). Различия состоят в следующем. Десятичное число не должно начинаться с нуля, предшествующий нуль означает, что это восьмеричное число, а предшествующие 0x (или 0X) означают шестнадцатеричное.

*Литералы с плавающей точкой* могут иметь следующие части: десятичное целое число, десятичная точка ('.'), дробная часть (десятичное целое), признак порядка ('e' или 'E') и порядок (десятичное целое со знаком). Некоторые примеры литералов с плавающей точкой - 3.1415, -3.1E1, .107e2, и 2E-10.

*Литералы Boolean.* Тип Boolean имеет два значения: true и false.

## Операции

*Операция присваивания* (=) присваивает значение выражения, стоящего справа от оператора =, переменной, стоящей слева от него, или значение правого операнда левому. То есть  $x = y$  присваивает переменной  $x$  значение  $y$ . Существует сокращенная форма оператора присваивания, общий вид которой:

$\langle \text{переменная} \rangle \langle \text{знак операции} \rangle = \langle \text{выражение} \rangle$

при этом подразумевается выполнение следующего оператора присваивания:  $\langle \text{переменная} \rangle = \langle \text{переменная} \rangle \langle \text{знак операции} \rangle \langle \text{выражение} \rangle$

Знаком операции могут быть: +(сложение), -(вычитание), \*(умножение), /(деление), %(деление по модулю), <<(сдвиг влево), >>(сдвиг вправо), >>>(сдвиг вправо с заполнением нулями), &(побитовое И), ^ (побитовое исключающее ИЛИ), |(побитовое ИЛИ). Из сказанного выше следует, что, например, оператор  $x+=2$ ; алгоритмически эквивалентен оператору  $x=x+2$ ;

*Операции сравнения* сравнивают операнды и возвращают логическое значение, основанное на истинности сравнения. Операнды могут быть числовыми или строковыми. Когда используются строковые значения операндов, сравнение проводится с помощью стандартного лексикографического анализа.

Операции сравнения: равно(==), не равно(!=), больше(>), больше или равно(>=), меньше(<), меньше или равно(<=)

Стандартные *арифметические операции* - сложение (+), вычитание (-), умножение (\*), деление (/), деление по модулю (%), инкремент (++), декремент (-- ) работают также, как и в языке C (C++). Одноместное отрицание, или унарный минус, (-) предшествует операнду и инвертирует его.

*Логические операции* должны иметь логические операнды и возвращают логический результат (табл.11.1).

Оператор	Использование	Описание
И	$a \& b$	возвращает true, если $a=\text{true}$ и $b=\text{true}$
ИЛИ	$a \mid b$	возвращает false, если $a=\text{false}$ и $b=\text{false}$
НЕ	$!a$	возвращает true, если $a=\text{false}$ и наоборот

Слияние строк осуществляется с помощью *оператора конкатенации*. Например, "two" + " words" возвращает значение "two words". Оператор присваивания += может также использоваться при конкатенации, например, если переменная mystring имеет значение "alpha" и вычисляется выражение mystring+="bet", то возвращается результат равный "alphabet", который присваивается переменной mystring.

### Организация разветвленных алгоритмов.

*Условный оператор.* Синтаксис: if (условие){ оператор1 [ ] } else { оператор2 [ ] }

Если условие истинно, выполняется оператор1, иначе - оператор2.

*Пример 11.1.* Пример стандартного оператора if:

```
if (a == 1) window.alert("1 найдено!");  
a=0;
```

Нельзя путать оператор сравнения (==) с оператором присваивания (=).

*Совместное использование логических и условных операторов.* Очень часто необходимо проверить сразу несколько условий. Это можно сделать с помощью логических операций || (логическое ИЛИ) и && (логическое И).

*Пример 11.2.*

```
if (phone == " ") window.alert("Ошибка! ")  
if (email == " ") window.alert("Ошибка! ")
```

С помощью логического оператора эти два оператора объединяются в один:

```
if (phone == " " \ email == " ") window.alert("Ошибка! ")
```

*Оператор else* - альтернативные действия оператора if, которые выполняются, если условие оператора if ложно.

*Пример 11.3.* Логическая переменная checkData получает значение true, если значение переменной data равно эталонному значению. В противном случае выводится сообщение об ошибке и значение checkData становится равным false.

```
etalon=10;  
if (data == etalon) { checkData =  
true;} else { alert("Произошла  
ошибка системы");  
checkData = false;  
}
```

*Оператор switch.* При необходимости выбрать более чем из двух вариантов, удобно пользоваться оператором множественного выбора switch.

Оператор switch состоит из нескольких основных элементов:

- Ключевое слово switch, за которым следует селектор выбора. Тело оператора switch заключается в фигурные скобки.

- Один или несколько вариантов, определяемых словом case. Каждая такая строка содержит значение, сравниваемое со значением селектора. Если значения совпадают, операторы этого варианта выполняются. Если нет - пропускаются. Ключевое слово break используется для определения конца действия текущего варианта.

- Необязательный оператор default выполняется, если ни один из вариантов не выполнен.

*Пример 11.4.*

```
switch (a)
{
    case 1:
        wmdow.alert('T'); break;
    case 2: window.alert('2');
        break;
    case 3:
        window.alert('3'); break;
    default:
        window.alert('Error!');}
```

### **Организация циклических алгоритмов.**

Цикл - автоматическое повторение выполнения одного или нескольких операторов, составляющих тело цикла.

*Оператор цикла for.* В операторе for цикл повторяется до тех пор, пока условие не станет ложным.

Синтаксис:

```
for ([начальное выражение];[условие];[выражение приращения]) {тело
                                                                цикла}
```

Внутри тела цикла могут использоваться операторы break и continue, соответственно для выхода из цикла и выхода из текущей итерации цикла.

*Пример 11.5.*

```
for (i=1; i<10; i++) { document.write('3ТО строка № ',i,'<br>');}
```

*Оператор цикла while.* Цикл while выполняется до тех пор, пока указанное условие является истинным. Если условие сразу ложно, то цикл выполняться не будет.

*Пример 11.6.*

```
count=0;
value=3;
while (count<10) { count+=value; }
```

*Оператор цикла do ... while.* Единственное отличие от while - условие расположено в конце кода цикла.

*Пример 11.7.*

```
count=0;
value=3;
do { count+=value; } while (count<10);
```

В отличие от цикла с предварительным условием, данный цикл будет выполняться по крайней мере один раз, так как условие находится после тела цикла.

## Функции

*Функция* - логически заверченный и определенным образом оформленный набор операторов, который выполняет некоторую задачу. Чтобы использовать функцию, необходимо сначала определить ее.

Определение функции состоит из ключевого слова `function`, за которым следуют:

- имя функции,
- в круглых скобках список аргументов функции, отделенных друг от друга запятыми,
- блок операторов JavaScript, заключенный в фигурные скобки.

Операторы могут включать в себя вызовы функций, определенных в текущем приложении. Различие между определением и вызовом функции традиционно для языков программирования. Определение функции просто называет функцию и задает выполняемые ею действия. Запрос функции исполняет указанные действия с фактическими параметрами. Лучше всего определять функции для страницы в разделе `HEAD` документа. В этом случае все функции будут определены прежде, чем показано содержание документа. Иначе, в то время как страница еще не полностью загружена, пользователь мог бы исполнить действие, которое вызывает еще не загруженную функцию, что привело бы к ошибке.

*Пример 11.8.* Простая функция: `function simplefun(str)  
{ document.write("<HR><P>" + str); }`

Эта функция получает строку `str` как аргумент, добавляет некоторые HTML-признаки к ней с помощью оператора конкатенации и выводит результат в текущем документе с помощью метода `write`.

Можно использовать любую функцию, определенную в текущей странице. Можно также использовать функции, определенные в других поименованных окнах и фреймах. Для того, чтобы функция выполнялась ее необходимо вызвать. Предположим, функция `simplefun` была определена в разделе `HEAD` документа, тогда выполнить ее можно, например, следующим образом:

```
<SCRIPT language="JavaScript"> simplefun("Вызвать функцию  
simplefun"); </SCRIPT>
```

Аргументы функции не ограничиваются строками и числами, можно передавать в качестве аргументов также и целые объекты. Функция может быть рекурсивной, то есть может вызывать сама себя. Рассмотрим функцию, которая вычисляет факториал.

*Пример 11.9.*

```
function factorial(n)
```

```
    { if (n<=1) { return 1; }
```

```
      else { result = (n *  
                    factorial(n-1)); return  
                    result;  
            }
```

```
    }  
    for (x = 0; x < 5; x++) { document.write("<BR>", x, " factorial is ",  
factorial(x)); }
```

Результат:

0 factorial is 1

1 factorial is 1  
2 factorial is 2  
3 factorial is 6  
4 factorial is 24  
5 factorial is 120

## Введение в Объектную Модель

С помощью объектной модели JavaScript взаимодействует с HTML страницей, загруженной в браузер, и может изменять и контролировать свойства окна просмотра и загруженной страницы.

В Объектной Модели браузера существуют два главных объекта:

- window - собственно окно просмотра,
- document - загруженная в это окно страница.

Чтобы увидеть это, следует загрузить в браузер какую-либо HTML-страницу и посмотреть на экран.

Все, что относится к браузеру: кнопки навигации, строка состояния, поле ввода адреса, меню и т.д. все это принадлежит объекту window.

Все, что относится к загруженной странице: графика, формы, ссылки, заголовок и т.д. - относится к объекту document.

Чтобы воспользоваться этими свойствами, получить к ним доступ, необходимо организовать ссылки на данные объекты. Например, чтобы записать что-либо в строку состояния нужно обратиться к соответствующему объекту status:

```
window.status = "Запись в строке состояния";
```

Объектная модель состоит из совокупности объектов, свойств, методов и событий.

Объект - это одна из частей браузера, которая выполняет определенные задачи. Например объект window отвечает за вид окна просмотра, за адрес, по которому загружен документ; объект document ответственен за загруженный документ; дочерний объект window - navigator в основном содержит информацию об агенте пользователя.

Свойства возвращают информацию о данном объекте. Например, через свойство объекта location можно узнать, какая страница загружена.

С помощью методов объекты выполняют возложенные на них функции. Например, методы объекта window позволяют открывать новые окна

Посредством событий объекты сообщают о каких-либо произошедших действиях. Например, событие для кнопки формы onClick сообщает, что кнопка была нажата.

## Объект window

К объекту window относится открытое окно браузера. Объект window является главным объектом объектной модели браузера, - все остальные объекты являются дочерними объектами объекта window.

Свойства window представлены в таблице 11.2:

Свойство	Описание
defaultStatus	Текст по умолчанию в строке состояния
document	Ссылка на объект окна document
frames	Ссылка на коллекцию frames, содержащую все фреймы окна
history	Ссылка на объект окна history
length	Количество фреймов в родительском окне
location	Ссылка на объект окна location
name	Название окна
navigator	Ссылка на объект окна navigator
parent	Возвращает родительское окно для текущего
self	Возвращает ссылку на текущее окно
status	Текст в строке состояния
top	Ссылка на главное окно

#### Методы window

- open С помощью метода open() открываются новые окна просмотра.

Синтаксис:

open( "иКБ\_открываемого\_окна", "Имя\_нового\_окна", [  
"Установки\_нового\_окна"] )

Значения свойства Установки нового окна (WindowFeatures) приведены в таблице.

Значение	Описание
directories	Создает панель ссылок браузера
fullscreen	Разворачивает окно полностью на весь экран
height	Задаёт высоту окна
location	Создаёт поле ввода адреса
menubar	Создаёт стандартное меню
resizeable	Определяет, сможет ли пользователь изменять размер окна
scrollbars	Создаёт полосы прокрутки
status	Создаёт строку состояния
toolbar	Панели инструментов браузера
width	Задаёт ширину окна

Самый простой способ открыть окно, это применение метода open без параметров. Нужно просто указать адрес открываемой страницы:

```
function Open_Window() { window.open("New.htm"); }
```

В результате на экране появится новое окно, размер которого зависит от настроек используемого браузера.

Можно задать параметры явно:

```
function Open_Window()  
{  
    window.open("New.htm","My_Win","toolbar=yes,height=200,width=  
600"); }
```

- close Закрывает текущее окно просмотра.

Пример 11.10.

```
<SCRIPT language="JavaScript">  
    function close_window() { close(); }  
</ SCRIPT >
```

Не следует злоупотреблять методом close, поскольку будучи вызванным он может закрыть окно браузера без каких-либо предупреждений. Последние версии браузеров Netscape Navigator и MS Internet Explorer выдают предупреждение пользователю, с помощью которого он может отменить закрытие текущего окна.

- alert Выводит на экран окно сообщения.

Синтаксис: alert (текст окна сообщения)

- confirm Выводит окно подтверждения.

Синтаксис: confirm (текст окна подтверждения)

- prompt Выводит окно ввода.

Синтаксис: prompt (текст окна ввода, начальное значение поля ввода)

Например, prompt (“Введите ваше имя”, “Вводите сюда”);

- setTimeout Выполняет заданное выражение по истечении установленного времени (в миллисекундах).

Синтаксис: setTimeout(expression, msec)

где *expression* - выражение, *msec* - задержка в миллисекундах.

### Объект location

Объект location содержит информацию об адресе страницы (URL) и позволяет управлять им.

- Свойство href объекта location содержит адрес страницы:  
alert(“Данная страница находится по адресу: “ + location.href);

• С помощью метода reload объекта location загруженная страница может быть обновлена. Эффект тот же, что и при нажатии кнопки Обновить.

Пример 11.11.

```
<SCRIPT language="JavaScript"> if (confirm(“Обновить страницу?”))  
    { location.reload();      //обновляем страницу:  
    alert(“Страница обновлена”);  
    }  
else { alert(“Страница не обновлена!”); }
```



</SCRIPT >

## Объекты Array

JavaScript не имеет типа данных, определяющих массив (array). Однако, можно использовать встроенный объект Array и его методы и работать с массивами в приложениях. Объект Array имеет методы для соединения, перевертывания и сортировки массивов. У него есть свойство для определения длины массива.

Создание массивов возможно двумя способами:

1. `arrayObjectName = new Array([arrayLength]);`
2. `arrayObjectName = new Array([element0, element1, ..., elementn]);`

`arrayObjectName` является или названием нового объекта, или свойством существующего объекта, `arrayLength` - начальная длина объекта. Можно получить доступ к этому значению используя свойство `length`; `element0, element1, ..., elementn` - список значений для элементов массива, если использована вторая форма инициализации массива, его длина определяется количеством аргументов.

Объект Array имеет следующие методы:

- `join` связывает все элементы массива в строку;
- `reverse` переворачивает элементы массива: первый элемент становится последним и наоборот;
- `sort` сортирует элементы массива.

*Пример 11.12.* Предположим, что мы определяем

следующий массив: `myArray = new`

`Array("Один", "Два", "Три");`

1. `myArray.join()` возвратит "Один,Два,Три";
2. `myArray.reverse` преобразует массив так, что `myArray[0]` есть "Три", `myArray[1]` есть "Два", и `myArray[2]` есть "Один".
3. `myArray.sort` сортирует элементы массива в лексикографическом порядке, так что `myArray[0]` есть "Два", `myArray[1]` есть "Один", и `myArray[2]` есть "Три".

Элементы массива можно определять с помощью оператора присваивания, например: `emp[1] = "Раз"; emp[2] = "Два"; emp[3] = "Три";`

Допускается инициализация массива: `myArray = new Array("Hello", myVar, 3.14159);`

*Пример 11.13.* Создаем двумерный массив и выводим на экран. `a = new Array(4);`

```
for (i=0; i < 4; i++) { a[i] = new Array(4);  
    for (j=0; j < 4; j++) { a[i][j] = "["+i+","+j+"]"; }  
    }  
for (i=0; i < 4; i++) { str = "Row "+i+":";  
    for (j=0; j < 4; j++) { str += a[i][j]; }  
    document.write(str, "<P>");  
    }
```

В результате будет напечатано:

```
Row 0:[0,0][0,1][0,2][0,3]  
Row 1:[1,0][1,1][1,2][1,3]  
Row 2:[2,0][2,1][2,2][2,3]  
Row 3:[3,0][3,1][3,2][3,3]
```

Можно обратиться к элементам массива, используя значение элемента или его индекс. Например, для массива `myArray = new Array("Один", "Два", "Три")`; обращение к первому элементу возможно как `myArray [0]` или `myArray ["Один"]`.

### Объект Date

Работа с датой и временем, установленными на компьютере, в JavaScript осуществляется с помощью объекта Date (табл.11.4): `var DateObject = new Date([значение]);`

где параметр *значение* может принимать следующие значения:

- миллисекунды - количество миллисекунд прошедшее от 01/01/70 00:00:00;
- год, месяц, день;
- год, месяц, день, часы, минуты, секунды;
- год, месяц, день, часы: минуты: секунды.

Метод	Описание
<code>getDate()</code>	Получить число месяца
<code>getDay()</code>	Получить номер дня недели
<code>getHours()</code>	Получить час
<code>getMinutes()</code>	Получить минуту
<code>getMonth()</code>	Получить номер месяца

<code>getSeconds()</code>	Получить секунду
<code>getTime()</code>	Получить количество миллисекунд между 1 января 1970 года и текущим временем
<code>getTimezoneOffset()</code>	Получить разницу в минутах между местным и гринвичским временем
<code>getYear()</code>	Получить год
<code>setDate()</code>	Установить день месяца
<code>setHours()</code>	Установить час
<code>setMinutes()</code>	Установить минуту
<code>setMonth()</code>	Установить месяц
<code>setSeconds()</code>	Установить секунду
<code>setYear()</code>	Установить год
<code>toGMTString()</code>	Преобразовать местное время во время по Гринвичу
<code>toLocaleString()</code>	Преобразовать время по Гринвичу в местное время
<code>UTC()</code>	Возвращает количество миллисекунд между 01/01/70 00:00:00 и заданным временем в формате объекта Date

Пример 11.14.

```
var D, day;
D = new Date();
```

```
day= D.getDate();
alert("Сегодня " + day + " число");
```

## Объект Math

Встроенный объект Math имеет свойства и методы для математических констант и функций. Например, свойство PI объекта Math имеет значение 3.14159..., который можно использовать как Math.PI. Стандартные математические функции являются методами Math (табл.11.5).

Методы объекта Math \_\_\_\_\_ Таблица 11.5

Метод	Описание
abs	абсолютное значение
sin, cos, tan	стандартные тригонометрические функции, аргумент в радианах
acos, asin, atan	обратные тригонометрические функции, возвращают значение в радианах
exp, log	экспонента и натуральный логарифм
ceil	возвращает наименьшее целое, большее или равное аргументу
floor	возвращает наибольшее целое, меньшее или равное аргументу
min, max	возвращает меньшее или большее (соответственно) из двух аргументов
pow	возведение в степень, первый аргумент основание, второй показатель степени
round	округляет аргумент до ближайшего целого
sqrt	квадратный корень

Например, если нужно использовать синус, следует писать Math.sin(1.56). Все тригонометрические методы Math используют аргументы в радианах.

Часто удобно использовать оператор with, когда есть раздел, в котором используется набор констант и методов, чтобы не повторять слово "Math":  
with (Math) { a = PI \* r\*r; y = r\*sin(alfa); x = r\*cos(alfa); }

## Объект document. Иерархия объектов страницы

Объект document содержит информацию о загруженном документе и отвечает за его отображение.

### Коллекции объекта document

- forms Коллекция forms представляет собой массив, куда входят все формы текущей страницы. Каждый объект form в свою очередь содержит массив elements, куда входят все объекты формы, т.е. объекты расположенные между тегами <FORM> и </FORM>.

- anchors Коллекция anchors представляет собой массив, в котором находятся все метки (то есть списки внутри страницы) документа.

Синтаксис anchor в HTML-странице:

```
<A name = "anchorName">anchorText</A>
```

- images Коллекция images содержит все рисунки, вставленные в документ, т.е. находящиеся в теге <IMG> , представляющие собой объект image. Каждый из них содержит коллекцию areas, содержащую выделенные области изображения. Свойство .src содержит url загруженного изображения

- links Коллекция links содержит все гиперссылки документа.

Доступ к коллекциям и элементам

JavaScript предоставляет несколько способов для доступа к ним.

1. Доступ через индекс массивов. Поскольку коллекции представляют собой массивы элементов, доступ к ним осуществляется как к элементам массивов. Так, если мы имеем массив links, первая встречающаяся гиперссылка в документе будет иметь индекс 0, вторая - 1, третья - 2 и т.д. Это справедливо для всех коллекций.

*Пример 11.15.* Страница, содержащая форму с полем ввода.

```
<FORM name="Form1">
<INPUT type="text" value="Вводить сюда" name="Text1">
<INPUT type="button" value="Показать" onClick="ShowText()">
</FORM>
```

Мы создали форму с полем ввода и кнопкой, вызывающей функцию ShowText(), которая будет показывать, какой текст введен в поле ввода. Для создания данной функции, нам нужно получить доступ к тексту, введенному в поле ввода. Делается это следующим образом. Известно, что все формы на странице входят в коллекцию forms, и каждая имеет свой порядковый номер. Наша форма единственная на странице, следовательно она будет первой, поэтому ее порядковый номер (индекс) будет равен 0. Поэтому, для получения доступа к нашей форме нужно составить следующую строку: document.forms[0].

После того, как мы получили доступ к форме, нам нужно получить доступ к полю ввода. Как упоминалось выше, все элементы, включенные в форму, доступны с помощью коллекции elements. Так как поле ввода встречается первым, то его индекс будет ноль. Таким образом получаем строку

```
document.forms[0].elements[0].
```

Данные, введенные в поле ввода, находятся в параметре value и полностью конструкция, содержащая данные строки ввода будет иметь следующий вид:

```
document.forms[0].elements[0].value.
```

2. Доступ через атрибут name

С помощью атрибута name каждой коллекции, а также ее элементам можно присвоить собственное имя для идентификации его сценарием.

Мы присвоили форме имя "Form1" и строке ввода имя "Text1".

Кнопка из сценария не вызывается, поэтому присваивать ей имя не обязательно.

Для получения доступа к введенным данным из сценария, заменим указание коллекций присвоенными именами:

```
document.Form1.Text1.value.
```

Свойства объекта document дают доступ к информации о загруженном документе (табл. 11.6).

Свойство	Описание
cookie	Выводит cookie, сохраненный браузером
lastModified	Дата последнего изменения страницы
location	Местонахождение страницы
parentWindow	Родительское окно (для фреймов)
referrer	Адрес страницы, из которой вызвана текущая страница
title	Заголовок документа
URL	Адрес страницы

Все эти свойства “только для чтения” т.е. их нельзя изменять из сценария.

С помощью свойства цвета объекта document (табл.11.7) мы можем получать и устанавливать значения цветового оформления страницы (фона, текста, гиперссылок и т.д.).

Свойства цвета. Таблица 11.7

Свойство	Описание
bgColor	Выдает/устанавливает значение фона страницы в RGB-представлении.
fgColor	Возвращает/устанавливает цвет обычного текста.
linkColor	Цвет гиперссылок.
alinkColor	Цвет активных гиперссылок.
vlinkColor	Цвет посещенных гиперссылок.

Пример 11.16:

```
<SCRIPT> function Change_bgcolor()
{ var Ccolor = prompt( "Введите новый цвет для фона страницы:", "" );
  document.bgColor = Ccolor;
}
</ SCRIPT >
```

При запуске данного сценария можно ввести значение цвета RGB-кодом, или его английским названием, например, green.

Методы объекта document (табл. 11.8) используются для работы с самим содержанием документа, для его изменения и редактирования.

Метод	Описание
open()	Открывает документ для записи
write()	Записывает данные в HTML-документ
close()	Закрывает документ
clear()	Удаляет содержимое страницы

- open Открывает HTML-документ для записи. Чтобы в документ что-либо записать, его нужно сначала открыть методом open(). Нельзя путать данный метод, с методом open() объекта window. Метод window.open() служит для открытия нового окна просмотра и может быть записан просто open(), а данный метод служит для открытия документа для записи и, чтобы было ясно, что это метод document, нужно обязательно ссылаться на него:  
document.open();

- write Позволяет записывать данные в документ.

Пример 11.17.

```
<HTML>
  <HEAD><TITLE>Использование метода write()</TITLE >
  </HEAD >
<BODY>
  <SCRIPT>
    document.write("Эта надпись будет видна только в браузерах,<^>
                      поддерживающих JavaScript.");
  </SCRIPT>
</BODY >
</HTML>
```

Как видно из примера, при помощи write() можно вставлять также тэги HTML. Вместе с методом write() существует также метод writeln(), который отличается тем, что в конце записываемой строки он автоматически добавляет в конце строки переход на новую строку. Эта разница практически незаметна, кроме нескольких случаев.

Пример 11.18.

Использование методов объекта document в элементе <PRE>, где сохраняется разбиение на строки:

```
<SCRIPT>
  document.writeln^PRE^"ТО первая строка");
  document.writeln("ТО вторая строка");
  document^r^ln^ "это..."); document.writeln("четвертая
  строка</PRE>");
</SCRIPT>
```

- close Метод close() закрывает открытую методом open() страницу. Как и метод open(), не следует данный метод смешивать с методом close() объекта window. Нужно всегда перед ним ставить ссылку на объект document: document.close(); так как иначе закроется все окно.

• **clear** Метод **clear**, применяется для очистки содержимого страницы, т.е. для удаления всего кода HTML. Для его использования необходимо сначала открыть страницу методом **document.open()**, а затем закрыть методом **document.close()**.

*Пример 11.19.*

```
<SCRIPT>
    function clear_doc()
    { if (confirm("Очистить документ?"))
      { document.open(); //открываем документ для
        редактирования document.clear(); //очищаем
        документ document.close(); //закрываем его
      }
    }
</SCRIPT>
```

### Включение JavaScript в HTML- документы

Существуют следующие способы подключения JavaScript-программ к HTML- документу:

1. Использование элемента **<SCRIPT>**.
2. Объявление JavaScript-файла, содержащего текст программы.
3. Объявление JavaScript-программы в качестве обработчика событий.

*Использование элемента SCRIPT.* Элемент **<SCRIPT>** может включать любое число JavaScript-операторов:

```
<SCRIPT> JavaScript - операторы ... </SCRIPT>
```

Документ может иметь любое количество элементов **SCRIPT**.

*Определение файла для JavaScript.* Атрибут **src** элемента **<SCRIPT>** позволяет определять файл как источник операторов JavaScript.

*Пример 11.20.*

```
<HEAD>
    <TITLE>My Page</TITLE>
    <SCRIPT src="myprog1.js">
        alert("Файл, содержащий скрипт, не подключился!");
    </SCRIPT>
</HEAD>
<BODY>

</BODY>
```

Внешние файлы JavaScript должны содержать только JavaScript-определения функций и операторы, в них не может быть HTML-тегов. Внешние файлы JavaScript должны иметь расширение **.js**.

*Пример 11.21.*

```
<HEAD>
    <SCRIPT language="JavaScript">
        <!-- Скрываем от старых браузеров
        function myfunc(number) { return number + number;}
        // Конец сокрытия -- >
    </SCRIPT>
```

```

</HEAD>
<BODY>
<SCRIPT language="JavaScript">
    document.write("Удвоенный аргумент = ",
        myfunc(7), ".");
</SCRIPT>
<P> OK !
</BODY>

```

Функция `myfunc` имеет один аргумент, названный `number`. Состоит она из одного оператора `return number+number`, который должен возвратить удвоенное значение аргумента функции. Вызывается функция с аргументом 7. Результат, выводимый на экран, выглядит следующим образом:

Удвоенный аргумент = 14.  
OK!

*Использование write-метода.* С помощью `write`-метода можно выводить информацию в окно навигатора так же, как и с помощью HTML-операторов, но `write`-метод обладает большими возможностями, например можно работать с переменными аргументами. По этим причинам, `write` - один из наиболее часто используемых JavaScript-методов. `write`-метод работает с любым числом строковых аргументов, которые могут быть строковыми литералами или переменными. Можно также использовать строковую конкатенацию - оператор ( `+` ), чтобы создать одну строку из нескольких.

*Пример 11.22.* Динамическое формирование HTML-страницы.

```

<HEAD>
<SCRIPT>
    <!-- Скрываем от старых браузеров // Эта функция выводит
    горизонтальную линию заданной ширины function bar(widthPct)
    { document.write("<HR align='left' width='widthPct' + '%'>");}

    // Эта функция показывает заголовок указанного уровня и
    некоторый текст function output(headLevel, headText, text)
    { document.write("<H", headLevel, ">", headText, "</H",
        headLevel, "><P>", text);
    }
    // Конец сокрытия-- >
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
    <!-- Начало сокрытия bar(25);
    output(2,"Это заголовок 2-го уровня",
        "Здесь можно добавить любую строку");
    // Конец сокрытия -- >
</SCRIPT>
<P> Это стандартный HTML, в отличие от предыдущей строки,
    которая сгенерирована.
</BODY>

```

Раздел `HEAD` этого документа определяет две функции:

-`bar` - функция выводит горизонтальную линию заданной ширины

-`output` - функция показывает заголовок указанного уровня и некоторый текст



Раздел BODY вызывает обе функции с фактическими параметрами.

Динамическое формирование таблицы:

```
var cols = prompt ('Введите количество строк', '');
var rows = prompt ('Введите количество столбцов', '');
cols = parseInt (cols);
rows = parseInt (rows);
document.write ('<table border="1" cellpadding="2" cellspacing="0"
width="90%">');
for (var r = 0, lr = rows + 1; r < lr; r++)
{
    for (var t, z = '', c = 0, lc = cols + 1; c < lc; c++)
    {
        t = r * c; if (!r && !c) t = '&nbsp;'; else if (!Math.min (r, c)) t =
Math.max (r, c);
        z += '<td style="background-color: ' + ((!r || !c) ? 'orange' :
'silver') + '">' + t + '</td>';
    }
    document.write ('<tr>' + z + '</tr>')
}
document.write ('</table>');
```

### 3. Групповое задание

Пользователь задаёт название конфет, цену за 1 кг, и вес покупки. Рассчитать стоимость покупки. Сформированные покупки с рассчитанной стоимостью должны добавляться в таблицу на странице.

Реализовать JavaScript в отдельном файле. Для оформления страницы использовать каскадные таблицы стилей, реализованные также в отдельном файле.

### 4. Варианты для индивидуальных заданий

- 1) Конвертер валют \$ ↔ руб (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).
- 2) Часы, показывающие время разных часовых поясов.
- 3) Конвертер расстояний ml ↔ км (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).
- 4) Генератор уникальных 8-значных паролей из букв и цифр.
- 5) Конвертер градусов °F ↔ °C (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).
- 6) Расчет Sin(x), Cos(x), Tan(x) и CoTan(x) для вводимого пользователем значения – x.
- 7) Конвертер длин " ↔ см (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).
- 8) Расчет площади и периметра различных двумерных фигур (результат расчета выводить в таблицу).
- 9) Конвертер веса пуд ↔ кг (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

10) Расчет объема различных трехмерных фигур (результат расчета выводить в таблицу).

11) Игра «Камень, ножницы, бумага» против компьютера до трех побед (выводить в таблицу статистику побед).

12) Конвертер мощности л.с  $\leftrightarrow$  кВт (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

13) Расчет дохода в зависимости от вложения в банк (результат расчета выводить в таблицу).

14) Разработать будильник при срабатывании которого начинает в случайном порядке меняться цвет фона страницы.

15) Конвертер кошачьего возраста чел. год  $\leftrightarrow$  кош. год (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

16) Пользователь отгадывает загаданное компьютером число, ориентируясь на подсказки «больше» или «меньше».

17) Конвертер градусов  $^{\circ}\text{K} \leftrightarrow ^{\circ}\text{C}$  (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

18) Отсчет количества дней до нового года по старому и новому стилю.

19) Конвертер веса золотая унция  $\leftrightarrow$  гр (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

20) Компьютер пытается отгадать число, загаданное пользователем ориентируясь на подсказки «больше» или «меньше».

21) Конвертер космических расстояний свет. год км (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

22) Разработать галерею картинок с возможностью их пролистывания.

23) Конвертер курса криптовалюты В  $\leftrightarrow$  руб (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

24) Конвертер времени сутки  $\leftrightarrow$  мин (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

25) Игра в кости против компьютера двумя кубиками (выводить в таблицу статистику побед).

26) Конвертер объемов информации Кбайт  $\leftrightarrow$  бит (сконвертированные значения должны добавляться в таблицу на странице после выполнения конвертации).

27) По введенным анкетным данным (имя героя, лучший друг героя, любимый цветок, любимое занятие и т.п.) сгенерировать в новом окне связанную сказку.

28) Бегущая строка с текстовым сообщением введенным пользователем.

29) Разработать карусель с изображениями.

30) Разработать стрелочные часы.