

Internet- технологии

ЛЕКЦИЯ №6
ЯЗЫК НАПИСАНИЯ СЦЕНАРИЕВ JAVASCRIPT

Содержание

- Введение в JavaScript.
- Основы синтаксиса. Переменные, типы, условия, циклы, массивы, функции.
- Размещение кода на HTML-странице.
- Объектная модель документа. Методы и свойства объекта window.
- Поиск и манипуляция элементами страницы.



Введение в JavaScript

JavaScript – прототипно-ориентированный сценарный язык программирования. Является реализацией (диалектом) языка **ECMAScript**. Широко применяется в браузерах как язык сценариев для управления элементами гипертекстовых страниц на стороне **клиента**, без перезагрузки страницы.

Структурно JavaScript включает в себя:

- **ядро (ECMAScript)**, описывает основной набор элементов языка, таких как типы данных, управляющие структуры, операторы и т.д.
- объектная модель браузера (Browser ObjectModel, BOM), браузер-специфичная часть языка, прослойка между ядром и DOM;
- объектная модель документа (Document Object Model или DOM).

Если рассматривать JavaScript в отличных от браузера окружениях, то BOM и объектная модель документа (DOM) могут не поддерживаться .



Введение в JavaScript. Области применения

Области применения JavaScript:

- манипулирование элементами страницы (меню, создание визуальных эффектов (анимация, ротаторы), выполнение несложных вычислений, валидация, условная генерация контента, поиск по данным страницы и т.д.);
- управлять поведением браузера;
- написание пользовательских скриптов, выполняемых браузером при загрузке страницы;
- AJAX;
- создание серверных приложений;
- браузерные операционные системы;
- букмарклеты.



Синтаксис JavaScript

Сценарий на языке JavaScript – набор команд, разделенных точкой запятой (**необязательно**), последовательно обрабатываемых интерпретатором.

Для оформления однострочных **комментариев** используются **//**, многострочные и внутристрочные комментарии обрамляются **/*** и ***/**. Справочный комментарий – стиль **JSDoc**.

В отличие от HTML, JavaScript чувствителен к **регистру** букв.

Переменные используются для хранения значений. В названиях переменных можно использовать **буквы, подчёркивание, арабские цифры**. Начинаться могут только с буквы латинского алфавита либо с символа подчеркивания (tes, _tes, tes2, res_1).

Переменные объявляются с помощью ключевого слова **var**. Если присвоить значение несуществующей переменной она будет автоматически создана.

```
var ex=123;
```

```
ex2=20; //Объявление без var создает глобальную переменную
```



Синтаксис JavaScript. Именованние переменных

Принятые правила именованния переменных:

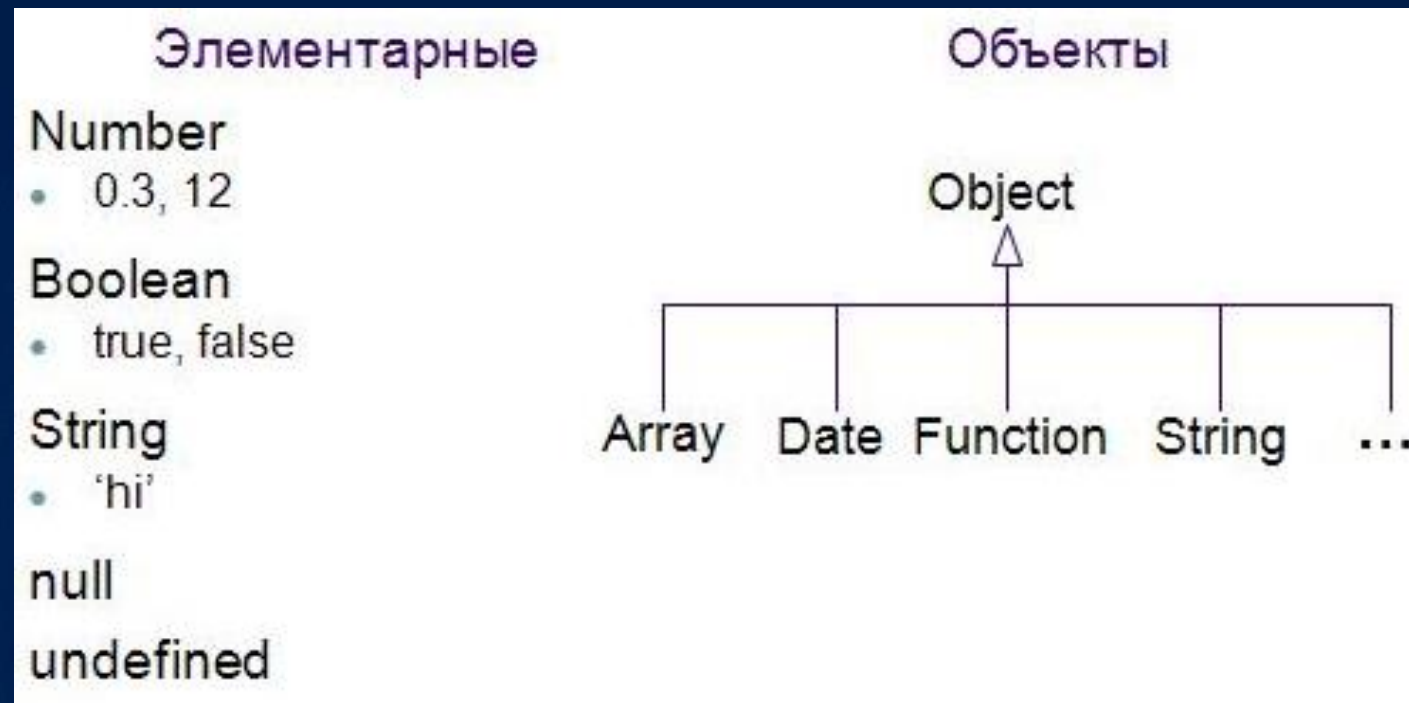
- имя переменной должно максимально соответствовать хранимым в ней данным. Имя переменной – **существительное**. Имя функции – **глагол** или начинается с глагола.
- избегать короткие имена.
- использование camelCase при составных именах переменных (borderLeftWidth).
- транслит недопустим.

Советы по стилю написания JavaScript-кода:

- Горизонтальный отступ, при вложенности – два(или четыре) пробела. Не табуляция!
- Для лучшей разбивки кода – не жалеете переводы строки.
- Фигурные скобки на той же строке.
- Длина строки кода – согласно используемым мониторам.



Основы синтаксиса. Типы данных



Переменные элементарных типов – создаются простым указанием данных:

```
var orange = "Апельсин";
```

Объекты создаются при помощи оператора new:

```
var ejik = new Animal ("ёжик"); var user = { name: "Вася" };
```



Основы синтаксиса. Типы данных. *typeof*

Оператор `typeof` возвращает тип аргумента. У него есть два синтаксиса:

`typeof x`

`typeof(x)`

Результатом `typeof` является строка, содержащая тип переданного оператору аргумента:

```
typeof undefined; // "undefined"
```

```
typeof 0; // "number"
```

```
typeof true; // "boolean"
```

```
typeof "foo"; // "string"
```

```
typeof {} // "object"
```

```
typeof null; // "object"
```

```
typeof function(){} // "function"
```



Основы синтаксиса. Типы данных. Число

Специальные числовые значения **Infinity** (бесконечность) и **NaN** (ошибка вычислений).

Например, бесконечность **Infinity** получается при делении на ноль:

```
alert( 1 / 0 ); // Infinity  
var x = Infinity;
```

Ошибка вычислений **NaN** будет результатом некорректной математической операции, например:

```
alert( "нечисло" * 2 ); // NaN, ошибка
```

Формы записи:

В шестнадцатичной системе:

```
alert( 0xFF ); // 255
```

Запись в «научном формате» - указав число разрядов, на которые сдвинуть влево или вправо:

```
alert( 3e5 ); // 300000      alert( 3e-5 ); // 0.00003
```



Основы синтаксиса. Типы данных. Строка

В JavaScript любые текстовые данные являются строками. Не существует отдельного типа «символ», который есть в ряде других языков. Нет разницы между двойными и одинарными кавычками. Если строка в одинарных кавычках, то одинарные кавычки внутри должны быть **экранированы** обратным слешем \

```
var text = "моя строка"; alert(text.length ); // 10
var anotherText = 'еще строка';
var str = '\I\'m a JavaScript programmer'; alert( str.indexOf("JavaScript ") ); // 6
var str = "012345"; alert( str.charAt(2) ); // "2" alert( str[0] ); // "0"
```

Строки могут содержать специальные символы:

\n – перевод строки; \r – возврат каретки; \t – табуляция;

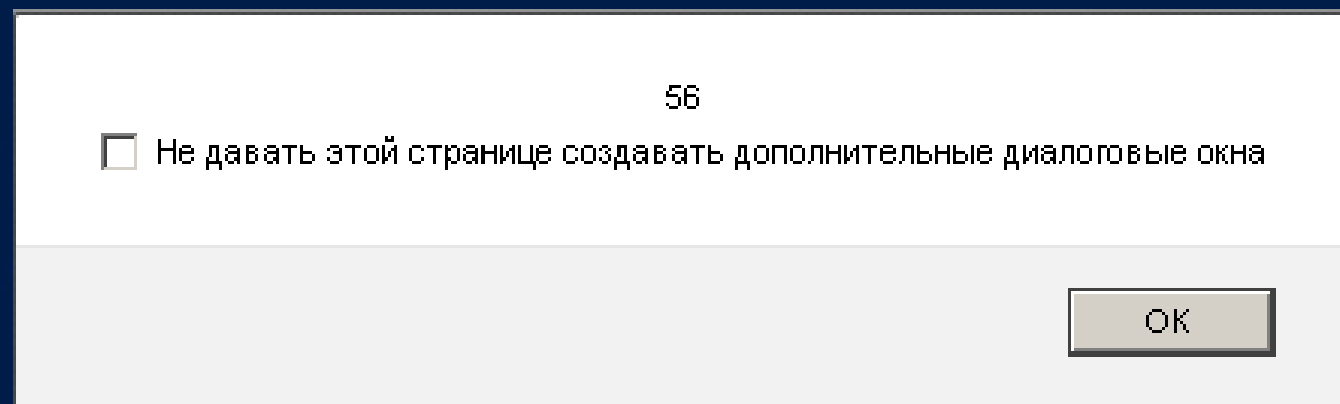
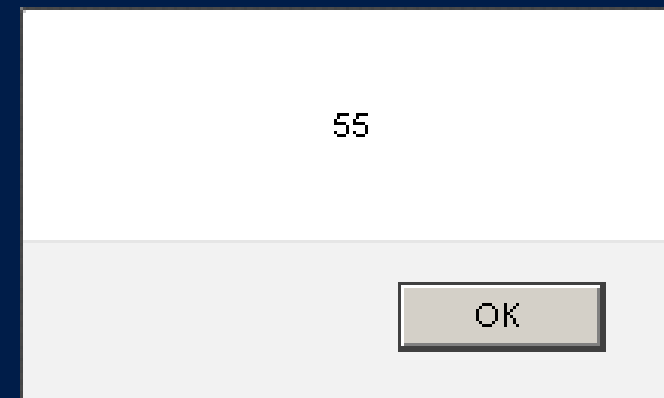
\uNNNN – символ в кодировке Юникод с шестнадцатеричным кодом 'NNNN'. Например, '\u00A9' - представление символа копирайт ©



Основы синтаксиса. Типы данных. Строка. Пример

```
var str = "JavaScript was originally developed in 10 days in May 1995";  
var target = "9"; // цель поиска
```

```
var pos = 0;  
while (true) {  
    var foundPos = str.indexOf(target, pos);  
    if (foundPos == -1)  
        break;  
    // нашли на этой позиции  
    alert( foundPos );  
    // продолжить поиск  
    // со следующей  
    pos = foundPos + 1;  
}
```



Основы синтаксиса. Приведение типов

JavaScript – это динамически типизированный язык. Типы данных преобразуются автоматически.

При наличии **бинарной** операции "+" числа преобразуются в строки. Для преобразования к **числу** в явном виде можно вызвать `Number(val)`, либо, что короче, поставить перед выражением **унарный** плюс "+".

Преобразование к **true/false** происходит в логическом контексте, таком как `if(value)`, и при применении логических операторов.

о, пустая строка, `null`, `undefined` и `NaN` – становятся **false**.

Остальное, в том числе и любые объекты – **true**.

Для явного преобразования используется двойное логическое отрицание **!!value** или вызов **Boolean(value)**.



Основы синтаксиса. Приведение типов. Примеры

+ "-12" // -12

+" \n34 \n" // 34, перевод строки \n является пробельным символом

+"" // 0, пустая строка становится нулем

+"1 2" // NaN, пробел посередине числа - ошибка

6 / '3' //2

6 / "3" //2

7 / 0 //Infinity

7 / '0' //Infinity

7 + 0 //7

6 + '3' //63

'37' + 6 //376

'44' - 2 //42

'2' * '3' //6

4 + 5 + 'px' //9px

'\$' + 4 + 5 //\$45

'4px' - 2 //NaN



Основы синтаксиса. Массивы

Каждому значению, которое заносится в массив присваивается уникальный идентификатор, по которому затем сможете обращаться к данному элементу внутри массива. Массивы в JavaScript не являются **ассоциативными** массивами. Для связывания ключей и значений в JavaScript есть только **объекты**.

Массив создается следующими способами:

```
item = new Array();  
item[0] = "Автомобиль"; item[1] = "Микроволновая печь"; item[5] = "Пылесос";  
  
var item = new Array("Автомобиль", "Микроволновая печь", "Пылесос");  
  
var item = ["Автомобиль", "Микроволновая печь", "Пылесос"];
```



Основы синтаксиса. Массивы

Для того, чтобы обратиться к элементу сохраненному в массиве необходимо указать **имя массива** и **индекс** желаемого элемента в квадратных скобках (массив[индекс]). Индексы элементов начинаются с нуля.

Свойство **length** позволяет узнать количество элементов в массиве. Присвоение свойству length меньшей величины урезает массив, однако присвоение большего значения не производит никакого эффекта.

```
var arr = ['первый элемент', 'второй элемент'];  
console.log(arr[0]);           // напечатает 'первый элемент'  
console.log(arr[1]);           // напечатает 'второй элемент'  
console.log(arr[arr.length - 1]); // напечатает 'второй элемент'
```



Основы синтаксиса. Функции

Синтаксис объявления функции:

```
function function-name( parameter-list ) {  
    //declarations and statements (Function body)  
}
```

Функция может содержать локальные переменные, объявленные через **var**. Такие переменные видны только внутри функции.

Пример:

```
var userName = 'Вася';  
function showMessage() {  
    var message = 'Привет, я ' + userName; // можно изменить  
    alert(message);  
}  
// вызов функции  
showMessage(); // Привет, я Вася
```



Основы синтаксиса. Функции

В своей простейшей форме функция представляет собой часть программного кода, который в любое время может быть вызван по его имени.

*глобальная
переменная*

*локальная
переменная*

```
var globalVar = 0;  
function factorial(n) {  
    if ((n == 0) || (n == 1))  
        return 1;  
    else  
        return n * factorial(n - 1);  
}  
function factorial2(n) {  
    var m = 1;  
    for (var i = 1; i <= n; i++)  
        m = m * i;  
    return m;  
}
```

*имя
функции*

*список
аргументов*

```
alert(factorial(5)); //120  
alert(factorial2(5)); //120
```



Основы синтаксиса. Встроенные функции

`parseInt(string , radix)`

`parseFloat(string)`

Функция `parseInt` возвращает целочисленное значение, получаемое в результате интерпретации содержимого строкового аргумента **string** как **числа** согласно указанному основанию системы счисления **radix**.

`isNaN(number)`

Возвращает `true`, если результат равен NaN, и `false` в прочих случаях.

`isFinite(number)`

Возвращает `false`, если результат равен NaN или бесконечность.

Функция `eval(code)` позволяет выполнить код, переданный ей в виде строки.

```
var a = 2;
```

```
eval('alert(a) '); // 2
```



Основы синтаксиса. Встроенные функции. Math

Все свойства и методы Math статичны.

Методы:

Math.exp

Math.min

Math.max

Math.random

Math.sqrt

Math.log

Math.round

Math.floor

Math.sin

Math.cos

Math.tan

Math.pow

Math.abs

Свойства:

Math.E

Math.PI

Генератор случайных чисел:

```
function getRandomInt(min, max)
{
    return Math.floor(Math.random()
        * (max - min)) + min;
}
```



Объектная модель документа (DOM)

Объектная модель – это представление HTML/XML документов в виде иерархически связанных объектов, характеризующихся наборами методов, свойств и событий.

Согласно DOM, документ (например, веб-страница) может быть представлен в виде дерева объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции:

- генерация и добавление узлов,
- получение узлов,
- изменение узлов,
- изменение связей между узлами,
- удаление узлов.



Объектная модель документа (DOM)

Каждый HTML-контейнер – это объект, который характеризуется свойствами, методами, событиями.

```
<A HREF=MenuInternetTech.html>InternetTech</A>  
document.links[0].href="MenuInternetTech.htm";
```

Методы

```
id=window.open("", "example", "width=400,height=150");
```

События

```
<input type=button value="Don't click here" onClick="window.alert('We repeate  
again: DON'T CLICK HERE');">
```

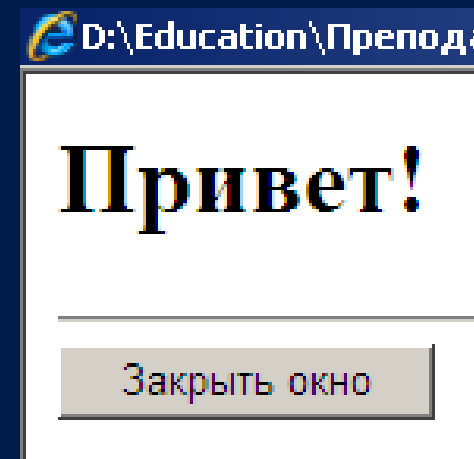


DOM. Методы

Методы объекта определяют функции изменения его свойств.

<script>

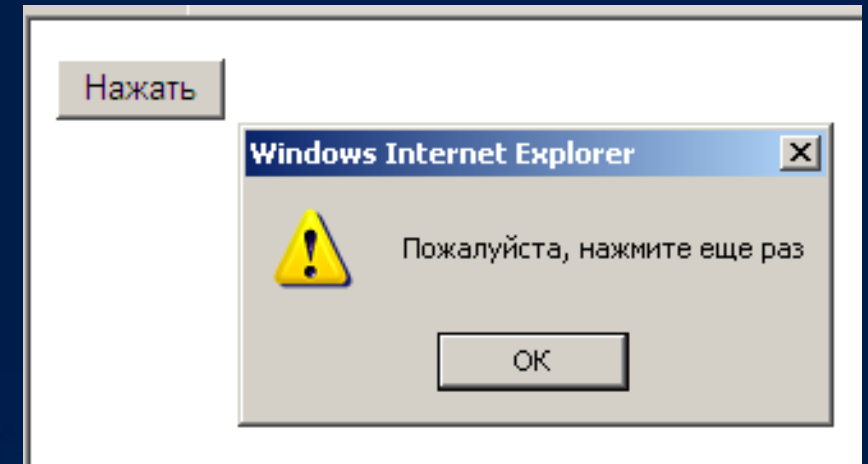
```
function hello() {  
  id=window.open("", "example", "width='400', height='150'");  
  id.focus();  
  id.document.open();  
  id.document.write("<h1>Привет!</h1>");  
  id.document.write("<hr>");  
  id.document.write("<input type='button' value='Заккрыть окно' ">");  
  id.document.write("onClick='window.opener.focus(); window.close();'>");  
  id.document.close();  
}  
</script>  
<BODY onload="hello()"></BODY>
```



DOM. События

```
<INPUT TYPE=button VALUE="Нажать" onClick="window.alert('Пожалуйста, нажмите еще раз');">
```

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



Размещение кода на HTML-странице

Исполняет JavaScript-код браузер. В него встроен интерпретатор JavaScript. Выполнение программы зависит от того, когда и как этот интерпретатор получает управление.

Способы размещения кода JavaScript на странице:

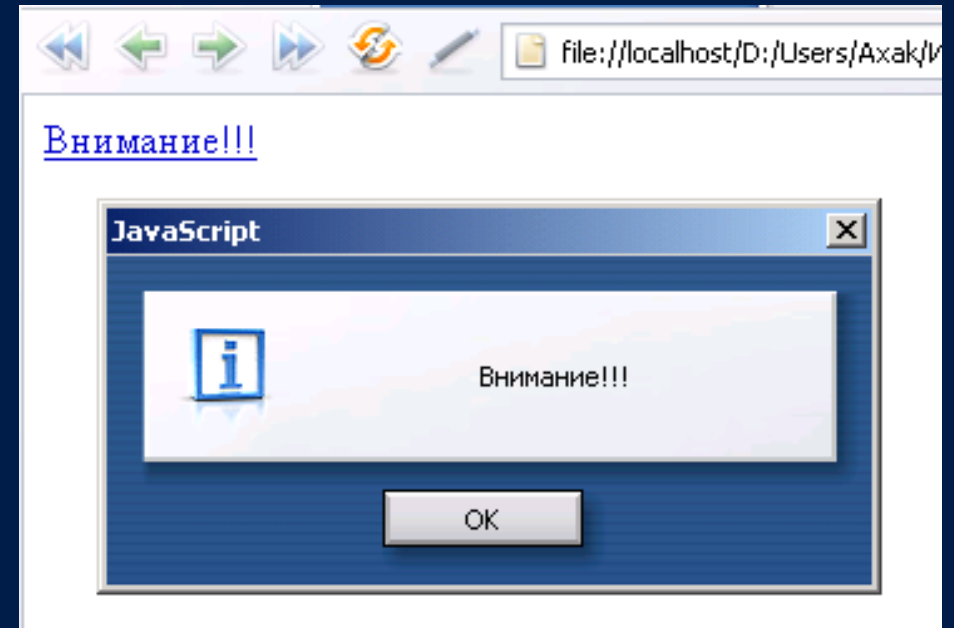
- гипертекстовая ссылка (схема URL);
- подстановка (entity);
- **обработчик события** (handler) – указываются в атрибутах контейнеров, с которыми эти события связаны.
- **вставка** (контейнер SCRIPT).



Размещение кода на HTML-странице. URL

URL указывают

- в атрибуте **HREF** контейнера A,
- в атрибуте **SRC** контейнера IMG,
- в атрибуте **ACTION** контейнера FORM
- и т.п.



` скрипт `

``

` Внимание!!!`



Размещение кода на HTML-странице. Подстановки

Подстановки (entity) поддерживаются только браузером Netscape Navigator 4.0. Подстановки имеют формат: &{код_программы}; и используются в качестве значений атрибутов HTML-контейнеров.

```
<SCRIPT>
function length() {
  str = "Hello!";
  return(str.length);
}
</SCRIPT>
<FORM>
<INPUT VALUE="Hello!" SIZE= "&{length()};">
</FORM>
```



Размещение кода на HTML-странице. Вставки

Для добавления JavaScript-кода на страницу можно использовать контейнер `<script></script>`, который рекомендуется помещать внутри контейнера `<head>`. Контейнеров `<script>` в одном документе может быть произвольное количество.

Атрибуты тега `<script>`:

- `language` – устанавливает язык программирования на котором написан скрипт. **Устарел.**
- `src` – адрес скрипта из внешнего файла для импорта в текущий документ.
`<script type="тип" src="URL"></script>`
- **type** – определяет тип содержимого `<script>`. Атрибут `type='text/javascript'` указывать необязательно, данное значение используется по умолчанию.
- `async` – загружает скрипт асинхронно.
- `defer` – откладывает выполнение скрипта до тех пор, пока вся страница не будет загружена полностью.



Размещение кода на HTML-странице. Пример

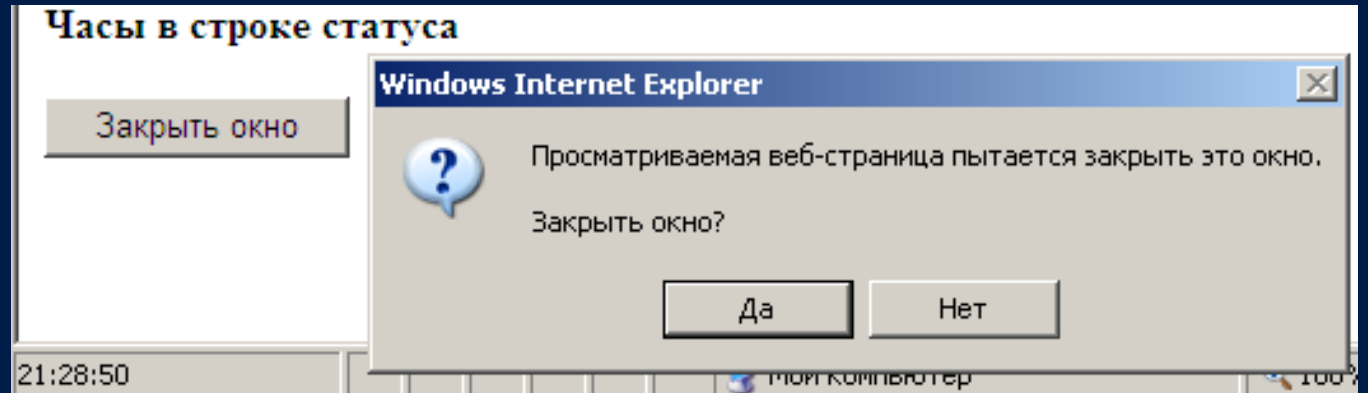
```
<body>
<script>
  document.write ('<table width="100%" border="1">');
  for (i=1; i<6; i++) {
    document.writeln("<tr>");
    for (j=1; j<6; j++)
      document.write("<td>" + i+j + "</td>");
    document.writeln("</tr>");
  }
  document.write ("</table> ");
</script>
</body>
```

11	12	13	14	15
21	22	23	24	25
31	32	33	34	35
41	42	43	44	45
51	52	53	54	55



Размещение кода на HTML-странице. Пример 2

```
<script>
function time_scroll() {
    d = new Date();
    window.status = d.getHours()
+ ":" + d.getMinutes()
+ ":" + d.getSeconds();
}
</script>
</head>
<body onLoad="setInterval(time_scroll, 1000)">
    <h4>Часы в строке статуса</h4>
    <input type="button" value="Заккрыть окно" onClick="window.close()">
</body>
```



Дата и время

Формат даты и времени: YYYY-MM-DDTHH:mm:ss.sssZ

Методы для получения
компонентов даты и
времени:

- `getDay()`,
- `getDate()`,
- `getMonth()`,
- `getFullYear()`,
- `getHours()`,
- `getMinutes()`,
- `getSeconds()`,
- **`getTime()`:**

Возвращает число миллисекунд, прошедших с 1 января 1970 года.

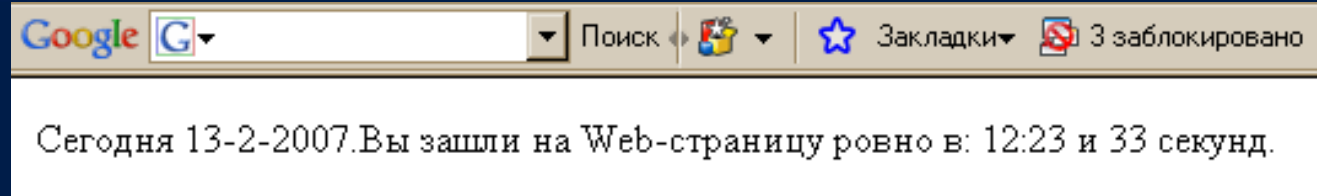
```
<SCRIPT LANGUAGE="JavaScript">
```

```
Now = new Date();
```

```
document.write("Сегодня " + Now.getDate()+  
"-" + Now.getMonth() + "-" + Now.getFullYear() + ".
```

```
Вы зашли на Web-страницу ровно в: " +  
Now.getHours() + ":" + Now.getMinutes() + " и " +  
Now.getSeconds() + " секунд.")
```

```
</SCRIPT>
```



Объектная модель документа



Структура HTML-документа отображается в иерархической структуре связанных объектов, соответствующих HTML тегам.



Объект Window

Window – это самый старший класс в иерархии объектов JavaScript. К нему относятся объект Window и объект Frame. Объект Window ассоциируется с окном программы-браузера, а объект Frame – с окнами внутри окна браузера, которые порождаются последним при использовании автором HTML-страниц контейнеров FRAMESET и FRAME.

Чаще всего используют следующие свойства и методы объектов типа Window:

Свойства:

status, location, history, navigator

Методы:

open()

close()

focus()



Объект Window. Управление окнами

Окно можно открыть (создать), закрыть (удалить), положить его поверх всех других открытых окон (передать фокус). Кроме того, можно управлять свойствами окна и свойствами подчиненных ему объектов.

наиболее популярные методы управления окнами:

`alert();`

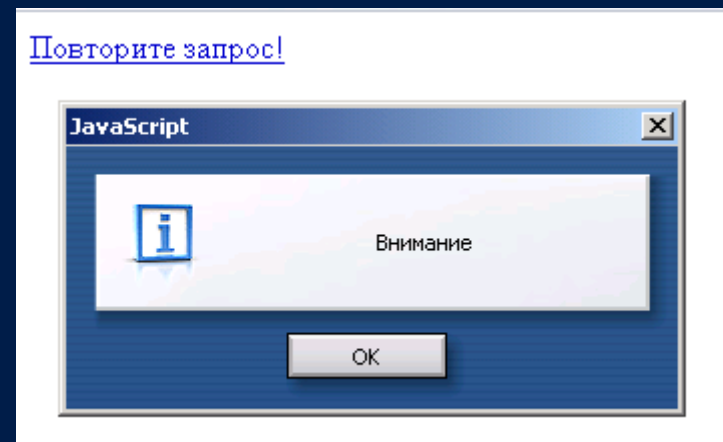
`confirm();`

`prompt();`

`open();`

`close();`

`focus();`



Управление окнами. *confirm()*

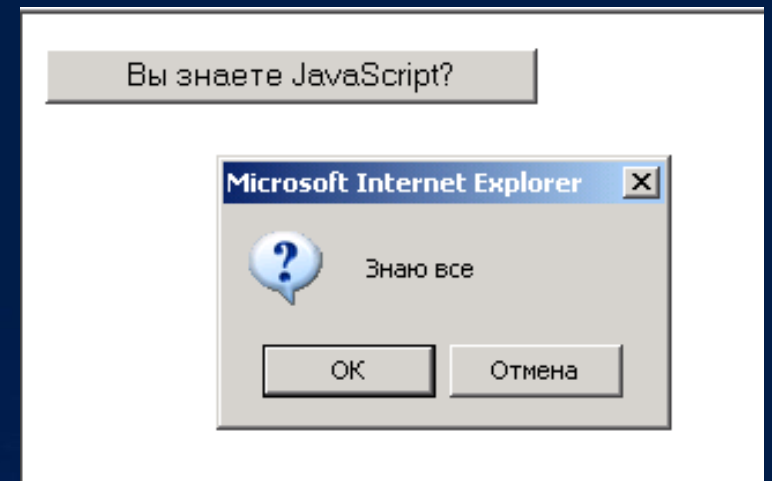
Метод `confirm()` позволяет задать пользователю вопрос, на который тот может ответить либо положительно, либо отрицательно:

```
<FORM>
```

```
<INPUT TYPE=button VALUE="Вы знаете JavaScript?"  
onClick = "if(window.confirm('Знаю все') == true) {  
document.forms[o].elements[o].value='Да'; } else {  
document.forms[o].elements[o].value='Нет'; };">
```

```
<BR>
```

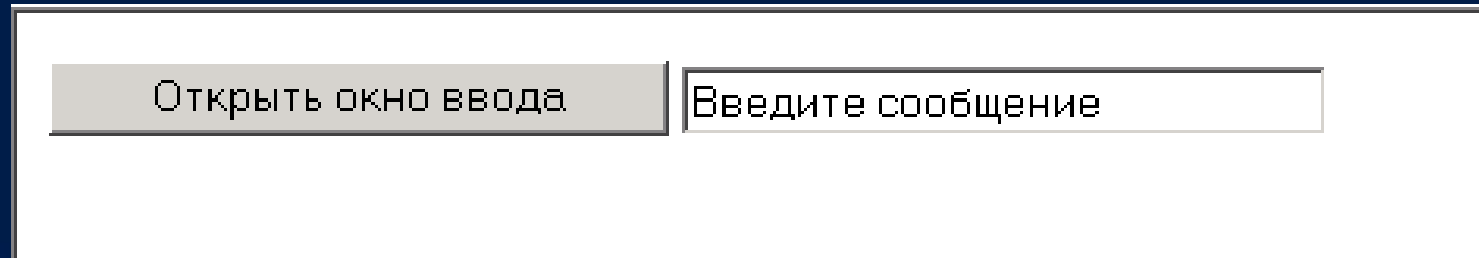
```
</FORM>
```



Управление окнами. *prompt()*

Метод `prompt()` позволяет принять от пользователя короткую строку текста, которая набирается в поле ввода информационного окна:

```
<FORM>  
<INPUT TYPE=button VALUE="Открыть окно ввода"  
onClick="document.forms[o].elements[1].value=window  
.prompt('Введите сообщение');">  
<INPUT SIZE=30>  
</FORM>
```



Открыть окно ввода

Введите сообщение



Управление окнами. *open()*

Метод *open()* предназначен для создания новых окон. В общем случае его синтаксис выглядит следующим образом:

```
open("URL","window_name","param,param,...", replace);
```

где: *URL* – страница, которая будет загружена в новое окно, *window_name* – имя окна, которое можно использовать в атрибуте *TARGET* в контейнерах *A* и *FORM*.

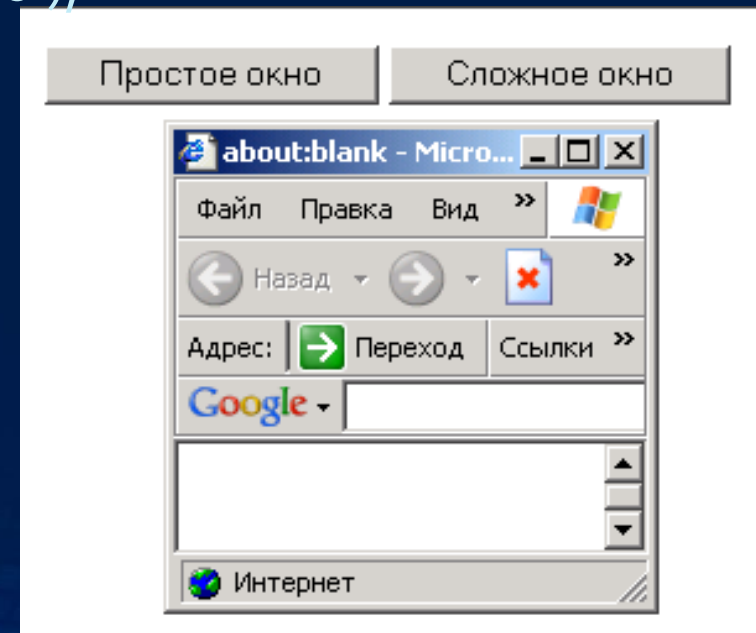
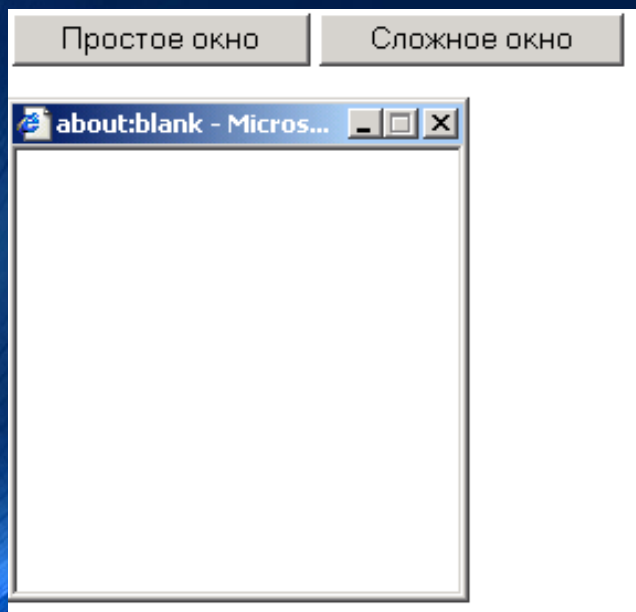
Параметры	Назначение
<i>replace</i>	Позволяет при открытии окна управлять записью в массив <i>History</i>
<i>Width, Height</i>	Ширина и Высота окна в пикселах
<i>Toolbar</i>	Создает <i>окно</i> с системными кнопками браузера
<i>Location</i>	Создает <i>окно</i> с полем <i>location</i>
<i>Status</i>	Создает <i>окно</i> с полем статуса <i>status</i>
<i>Menubar</i>	Создает <i>окно</i> с меню
<i>Scrollbar</i>	Создает <i>окно</i> с полосами прокрутки
<i>Resizable</i>	Создает <i>окно</i> , размер которого можно будет изменять



Управление окнами. Пример

```
<INPUT TYPE=button VALUE="Простое окно" onClick = "window.open('about:blank', 'test1','directories=no, height=200, location=no, menubar=no, resizable=no, scrollbars=no, status=no, toolbar=no, width=200');">
```

```
<INPUT TYPE=button VALUE="Сложное окно" onClick = "window.open('about:blank', 'test2', 'directories=yes, height=200, location=yes, menubar=yes, resizable=yes, scrollbars=yes, status=yes, toolbar=yes, width=200');">
```



Управление окнами. *close()*

Метод `close()` позволяет закрыть окно.

Если необходимо закрыть текущее, то:

```
window.close();
```

```
self.close();
```

Если необходимо закрыть родительское окно, т.е. окно, из которого было открыто текущее, то:

```
window.opener.close();
```

Если необходимо закрыть произвольное окно, то тогда сначала нужно получить его идентификатор:

```
id=window.open(...);
```

```
...
```

```
id.close();
```



Манипуляция элементами страницы

Для того, чтобы изменить узел DOM, например, изменить его свойства, добавить содержимое, нужно сначала его получить. Доступ к элементам DOM начинается с объекта `document`. Оттуда можно добраться до любых других узлов.

```
<body>
  <div>Пользователи:</div>
  <ul> <li>Маша</li>   <li>Вовочка</li> </ul>

  <script>
    var childNode = document.body.childNodes[1];
    for(var i=0; i<childNodes.length; i++) {
      document.write(childNode.childNodes[i].childNodes[0].nodeValue);
    }
  </script>
</body>
```

Пользователи:

- Маша
- Вовочка

Маша Вовочка



Манипуляция элементами страницы. Таблицы

У таблиц есть дополнительные свойства для более удобной навигации по ним:

table.rows – список строк TR таблицы.

tbody.rows – список строк TR секции.

tr.cells – список ячеек TD/TH.

tr.sectionRowIndex – номер строки в текущей секции THEAD/TBODY.

tr.rowIndex – номер строки в таблице.

td.cellIndex – номер ячейки в строке.

```
<table>
```

```
<tr><td> один</td><td> два</td></tr> <tr><td> три</td><td> четыре</td></tr>
```

```
</table>
```

```
<script>
```

```
var table = document.body.children[0];
```

```
alert(table.rows[0].cells[0].innerHTML); // "один"
```

```
</script>
```



Манипуляция элементами страницы

Самый удобный способ найти элемент в DOM – это получить его по id. Для этого используется вызов **document.getElementById(id)**

Следующий способ - получить все элементы с определенным тегом с помощью метода **document.getElementsByTagName(tag)**. Возвращает массив из элементов, имеющих заданный тег.

Метод **document.getElementsByName(name)** возвращает все элементы, у которых имя (атрибут name) равно данному.

Все методы **Elements** возвращают список узлов.

Метод **querySelectorAll** позволяет выбирать элементы с помощью CSS3-селектора.



Манипуляция элементами страницы

Самый удобный способ найти элемент в DOM – это получить его по id. Для этого используется вызов **document.getElementById(id)**

Следующий способ - получить все элементы с определенным тегом с помощью метода **document.getElementsByTagName(tag)**. Возвращает массив из элементов, имеющих заданный тег.

Метод **document.getElementsByName(name)** возвращает все элементы, у которых имя (атрибут name) равно данному.

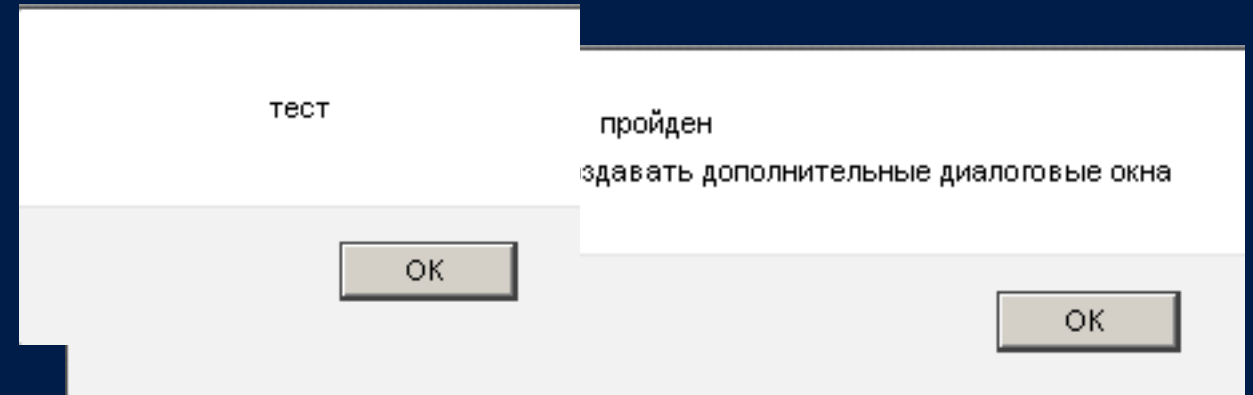
Все методы **Elements** возвращают список узлов.

Метод **querySelectorAll** позволяет выбирать элементы с помощью CSS3-селектора.



Манипуляция элементами страницы

```
<ul>
  <li>Этот</li>
  <li>тест</li>
</ul>
<ul>
  <li>полностью</li>
  <li>пройден</li>
</ul>
<script>
  var elements = document.querySelectorAll('ul > li:last-child');
  for(var i=0; i<elements.length; i++) {
    alert(elements[i].innerHTML ); // "тест", "пройден"
  }
</script>
```



Вопросы

- Какими возможностями обладает JavaScript?
- Для чего используются букмарклеты?
- Какие элементарные типы данных присутствуют в языке JavaScript?
- Правила приведения к логическому типу.
- Чем характеризуется объектная модель применительно к JavaScript?
- Какие свойства и методы вы знаете у объектов типа Window?
- Каким образом можно использовать объект Location?
- Какие методы управления окнами вы знаете?
- Способы получения элементов страницы.

