Focused forward

Netcracker

Лекция 4.

Введение в JavaScript

Web System Design and Development

План лекции

- JS: понятие, применение
- Структура и синтаксис
- Типы данных
- Методы и свойства
- Операции
- Операторы
- Циклы
- DOM – объектная модель документа
- Стратегии поиска HTML-элементов

JS: понятие, применение

- **прототипно-ориентированный сценарный язык программирования**
- **скриптовый язык, предназначенный для создания интерактивных веб-страниц**
- **не имеет никакого отношения к языку Java. Java разработан фирмой SUN. JavaScript - фирмой Netscape Communication Corporation. Первоначальное название - LiveScript.**

JS: понятие, применение

Расположение JavaScript кода внутри страницы

```
<!DOCTYPE HTML>
<html>
<body>
  <p>Начало документа...</p>

  <script>
    alert('Привет, Мир!');
  </script>

  <p>...Конец документа</p>
</body>
</html>
```

Когда браузер читает HTML-страничку, и видит <script> - он первым делом читает и выполняет код, а только потом продолжает читать страницу дальше.

JS: понятие, применение

Расположение JavaScript кода в заголовке HEAD

```
<html>
    <head>
        <script type="text/javascript">
            function count_rabbits() {
                for(var i=1; i<=3; i++) {
            alert("Из шляпы достали "+i+" кролика!")
            // оператор + соединяет строки
                }
            }
        </script>
    </head>
    <body>
        <input type="button" onclick="count_rabbits()"
        value="Считать кролей!"/>
    </body>
</html>
```

JS: понятие, применение

Расположение JavaScript кода во внешнем файле

```
<script type="text/javascript" src="/Путь/к/script.js"></script>
```

```
<html>
  <head>
<script src="rabbits.js"></script>
  </head>
  <body>
    <script>
       count_rabbits();
    </script>
  </body>
</html>
```

Содержимое файла rabbits.js:

```
function count_rabbits() {
    for(var i=1; i<=3; i++) {
        alert("Кролик номер "+i)
    }
}
```

JS: понятие, применение

Ограничения JavaScript

Большинство возможностей JavaScript в браузере ограничено текущим окном и страницей.

JavaScript не может читать/записывать произвольные файлы на жесткий диск, копировать их или вызывать программы. Он не имеет прямого доступа к операционной системе.

JavaScript, работающий в одной вкладке, почти не может общаться с другими вкладками и окнами. За исключением случая, когда он сам открыл это окно или несколько вкладок из одного источника.

Структура и синтаксис

Эти два примера кода эквивалентны:

```
return
result;
```

```
return;
result;
```

Но это не то же самое, что:

```
return result
```

Чтобы многострочные операторы работали как надо - перенос строки можно указывать обратным слешем \:

```
var a = “длинная \
  строка”
return \
  result;
```

Структура и синтаксис

Блоки

Задаются **фигурными скобками**. В отличие от C и Java, сам по себе блок не задает отдельную область видимости.

```
var i = 0
{
  var i=5
alert(i) // 5
}
alert(i) // тоже 5
```

```
// однострочные
комментарии

/*
многострочные
комментарии
*/
```

Структура и синтаксис

Директива **var** при объявлении переменной делает ее **локальной**, то есть видимой только внутри текущей функции. Все остальные переменные являются глобальными.

```
var a;
```

При объявлении можно тут же присвоить переменной значение и объявить другие переменные:

```
// то же самое, что 3 отдельных объявления с var
var a=5, b=6, str = "Строка "
```

Структура и синтаксис

Именование в JavaScript:

Имя может состоять из: букв, цифр, символов $ (может быть только первым символом в имени переменной или функции) и _

Первый символ не должен быть цифрой

В качестве имени переменной нельзя использовать ключевые слова JavaScript.

Имена переменных и функций чувствительны к регистру

Структура и синтаксис

Именование в JavaScript:

Underscore:	**first_name, last_name, master_card, inter_city**
Upper Camel Case (Pascal Case):	**FirstName, LastName, MasterCard, InterCity**
Lower Camel Case:	**firstName, lastName, masterCard, interCity**

https://www.w3schools.com/js/js_conventions.asp

Структура и синтаксис

Область видимости глобальных и локальных переменных

Внутри тела функции локальная переменная имеет преимущество перед глобальной переменной с тем же именем. Если объявить локальную переменную или параметр функции с тем же именем, что у глобальной переменной, то фактически глобальная переменная будет скрыта.

```
var x = "глобальная";   //Объявление глобальной переменной

function checkscope() {
var x = "локальная";  //Объявление локальной переменной с тем же именем
  document.write(x);    //Используется локальная переменная, а не глобальная
}

checkscope();
```

Структура и синтаксис

```
a = 1
function go() {
    var a = 6
}
go()
alert(a) // => 1
```

Переменная **a** в функции **go** объявлена как локальная. Поэтому глобальное значение **a=1** не изменяется в процессе выполнения **go**.

Структура и синтаксис

Функции

Можно создать два вида функций - **именованные** и **анонимные**.
Именованная функция **видна везде**, а **анонимная - только после объявления**.

Именованные	Анонимные

function имя(параметры) {
...
}
	var имя = function(параметры) {
	…
	}
	или:
	var имя = new Function(параметры, '...')

var a = sum(2,2)
function sum(x,y) {
return x+y
}
	var a = sum(2,2)
	var sum = function(x,y) {
	return x+y
	}

//функция sum определена ниже	//будет ошибка, т.к sum еще не существует

Структура и синтаксис

Пример определения и вызова именованной функции:

```
//определение функции
function starline() {
  for(var i = 0; i < 45; i++){
    document.write("*");
  }
  document.write("<br>");
}
starline();  //вызов функции
document.write("<p>это абзац</p>");
starline();
```

Структура и синтаксис

Параметры и аргументы функции

- Параметры указываются в определении функции внутри круглых скобок и являются ее локальными переменными, т.е. видны только в ее теле.
- При вызове функция может получать аргументы, которыми инициализируются параметры.

```
function getstr(car, place) {  //параметры car и place
  document.write('Моя машина - ' + car + ' и я еду на ней ' + place);
}
 function showbook(title) {
  str = '<p>книга называется: "' + title + '"</p>';
  document.write(str);
}
var house = "на дачу";
 getstr('BMW', house);  //аргументы - строковой литерал и переменная
showbook('Машина времени');
```

Структура и синтаксис

```
function f1(a,b,...)
{
/* тело функции*/
}
```

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction()
{
alert("Hello World!");
}
</script>
</head>

<body>
<button onclick="myFunction()">Try it</button>
</body>
</html>
```

Структура и синтаксис

Переменные

Переменные в JavaScript слабо типизированы, т.е. при объявлении **не нужно указывать тип.**

Можно присвоить любой переменной любое значение. При этом интерпретатор JavaScript (браузер) всегда знает, какого типа значение содержит данная переменная, а значит - какие операции к ней применимы.

Переменную не обязательно объявлять. Достаточно ее просто присвоить:

```
x = 5;
```

Имена переменных должны начинаться с букв или знаков, таких как $ и _

Имена переменных **чувствительны к регистру** (x и X – разные переменные).

Типы данных

- JavaScript является нетипизированным языком, **тип данных** для конкретной переменной при ее объявлении **указывать не нужно**.
- Тип данных переменной зависит от значений, которые она принимает.

String	Строковый тип
Number	Числовой тип: целые числа (*integer*) и числа с плавающей точкой (*floating-point number*)
Boolean	Логический тип: *true* (истина), *false* (ложь)
Object	Структура данных, в которой можно хранить любые данные в формате ключ-значение
Undefined and Null	Значение переменной без значения и пустое значение соответственно

Типы данных

String

Используется для хранения текста и манипуляций с ним.

```
var name="Sumy State University";
var name='Sumy State University';
```

Можно получить доступ к любой букве в строке, зная ее позицию:

```
var character=name[7];
```

Number

Все числа хранятся в формате **float64**, т.е. 8 байт с плавающей точкой. В качестве разделителя используется **точка**.

```
var x1=34.00;
var x2=34;
```

Типы данных

Boolean

Используется для преобразования в логическое значение (истина или ложь) значений других типов.

Чтобы привести значение к булевому:

- либо явное указание типа:

Boolean(a)

- либо двойное отрицание:

!!a

Типы данных

Object

Для **массивов, ключи которых являются строками**, используется **Object**.

Объект ограничен фигурными скобками. В фигурных скобках свойства объекта определяются как пары имен и значений **(имя: значение).** Свойства разделяются запятыми:

```
var person={firstname:"John", lastname:"Doe", id:5566};
```

Пробелы и разрывы строк не важны. Объявление объекта может быть таким:

```
var person={
firstname : "John",
lastname  : "Doe",
id          :  5566
};
```

Для получения свойства объекта используется оператор-аксессор: **точка** либо **квадратные скобки.**

```
name=person.lastname;
name=person["lastname"];
```

```
var key = "str"
alert(obj[key])
```

Квадратные **скобки** позволяют получать **свойство по переменной-ключу**.

Типы данных

Object

Для **массивов, ключи которых являются строками**, используется **Object**.

Объекты в JavaScript сочетают в себе два важных функционала:

- Первый – это **ассоциативный массив**: структура, пригодная для хранения любых данных.
- Второй – языковые **возможности для объектно-ориентированного программирования**.

Ассоциативный массив – структура данных, в которой можно хранить любые данные в формате **ключ-значение**.

Два способа объявления:

```
1. o = new Object();
2. o = {}; // пустые фигурные скобки
```

Типы данных

Object

Объект может содержать в себе любые значения, которые называются *свойствами объекта*.

В фигурных скобках свойства объекта определяются как пары имен (ключей) и значений **(ключ: значение).** Свойства разделяются запятыми:

Пробелы и разрывы строк не важны. Объявление объекта может быть таким:

```
var person={firstname:"John", lastname:"Doe", id:5566};
```

Для получения свойства объекта используется оператор-аксессор: **точка** либо **квадратные скобки.**

```
var person={
firstname : "John",
lastname  : "Doe",
id              :  5566
};
```

Квадратные **скобки** также позволяют получать **свойство по ключу**:

```
name=person.lastname;
name=person["lastname"];
```

Типы данных

Object

Основные операции с объектами – это **создание, получение и удаление свойств.**

Создадим объект **person** для хранения информации о человеке:

```
1 var person = {}; // пока пустой
```

Запишем содержимое:

```
person.name = 'Вася'

person.age = 25; //
```

Удаление осуществляется оператором delete:

```
1 delete person.age;
```

Чтобы прочитать их – также обратимся через точку:

```
1 alert( person.name + ': ' + person.age ); // "Вася: 25"
```

В JavaScript можно обратиться к любому свойству объекта, даже если его нет.
Резельтат: undefined

Типы данных

Array – разновидность Object

Для создания **массива с числовыми индексами** обычно используется объект типа **Array**.

```
var arr = new Array();
arr[0]= "my“;
arr[1]= "array“;
```

Часто используется **компактная форма записи**:

```
arr = new Array("my", "array")
alert(arr.length)  // =2
```

Свойство **length** содержит длину массива, которая всегда **равна последнему индексу плюс один.**

Также используется другая, почти эквивалентная, запись массива:

```
arr = [ "my", "array" ]
alert(arr[0]) // "my" <- нумерация от нуля
```

Типы данных

Undefined and Null

Undefined - это **значение переменной без значения**, т.е. её значение не было определено:

```
function a() {
alert('test')
}
result = a() // result будет undefined
alert(result === undefined) // true
```

Переменные можно сделать **пустыми** путем установки **значения NULL**.

```
cars=null;
person=null;
```

Типы данных

```
var x1 = {};           // new object
var x2 = "";           // new primitive string
var x3 = 0;            // new primitive number
var x4 = false;        // new primitive boolean
var x5 = [];           // new array object
var x6 = /()/;         // new regexp object
var x7 = function(){}; // new function object
```

Используйте **{}** вместо **new Object()**

Используйте **""** вместо **new String()**

Используйте **0** вместо **new Number()**

Используйте **false** вместо **new Boolean()**

Используйте **[]** вместо **new Array()**

Используйте **/()/** вместо **new RegExp()**

Используйте **function (){}** вместо **new Function()**

Вывод результата

into an HTML element - **innerHTML**

into the HTML output - **document.write()**

into an alert box - **window.alert()**

into the browser console – **console.log()**.

Методы и свойства

- Все значения в JavaScript, за исключением `null` и `undefined`, содержат **набор вспомогательных функций и значений, доступных «через точку».**
- Такие функции называют «**методами**», а значения – «**свойствами**».

Свойство length

У строки есть *свойство* `length`, содержащее длину:

```
1 alert( "Привет, мир!".length ); // 12
```

Можно и записать строку в переменную, а потом запросить её свойство:

```
1 var str = "Привет, мир!";
2 alert( str.length ); // 12
```

Все методы и свойства: https://www.w3schools.com/jsref/dom_obj_all.asp

Методы и свойства

Вызов метода – через круглые скобки!

Метод toFixed(n)

Округляет число до n знаков после запятой, при необходимости добивает нулями до данной длины и возвращает в виде строки (удобно для форматированного вывода).

```
var n = 12.345;

alert( n.toFixed(2) ); // "12.35"
alert( n.toFixed(0) ); // "12"
alert( n.toFixed(5) ); // "12.34500"
```

Методы и свойства

innerHTML

The innerHTML property sets or returns the HTML content (inner HTML) of an element.

Change the HTML content of a <p> element with id="demo":

```
document.getElementById("demo").innerHTML = "Paragraph changed!";
```

value

The value property sets or returns the value of the value attribute of a text field.
The value property contains the default value OR the value a user types in (or a value set by a script).

Change the value of a text field:

```
document.getElementById("myText").value = "Johnny Bravo";
```

Методы и свойства

style

The style property returns a CSSStyleDeclaration object, which represents an element's style attribute.
The style property is used to get or set a specific style of an element using different CSS properties.

Add a red color to an <h1> element:

```
document.getElementById("myH1").style.color = "red";
```

Все методы и свойства: https://www.w3schools.com/jsref/dom_obj_all.asp

Операции

В JavaScript используется несколько типов операций: **арифметические, присваивания, сравнения, логические и поразрядные (побитовые).**

Присваивание

Операция присваивания выглядит как знак равенства – “**=**”. Она присваивает значение, стоящее с правой стороны от нее, переменной, стоящей с левой стороны.

```
var x = 20;
var y = x + 32;
```

У операции присваивания имеется одна интересная особенность: она **позволяет создавать цепочку операций присваивания.**

```
var a, d, f;
a = d = f = 101;
```

Операции

Арифметические операции

Операция	Знак	Ее функция
Сложение	+	Сложение двух значений
Вычитание	-	Вычитание одного из другого
Умножение	*	Перемножение двух значений
Деление	/	Деление одного значения на другое
Получение остатка от деления	%	Деление одного значения на другое и возвращение остатка (деление по модулю)
Инкремент	++	Сокращенная запись добавления 1 к числу
Декремент	--	Сокращенная запись вычитания 1 из числа
Унарное отрицание	-	Превращение положительного числа в отрицательное или отрицательного в положительное

Операции

Операции сравнения

Операция	Символ	Функция
Равенство	==	Возвращает true, если операнды по обе стороны от операции равны друг другу
Неравенство	!=	Возвращает true, если операнды по обе стороны от операции не равны друг другу
Больше	>	Возвращает true, если операнд слева от операции больше, чем операнд справа от операции
Меньше	<	Возвращает true, если операнд слева от операции меньше, чем операнд справа от операции
Больше или равно	>=	Возвращает true, если левый операнд больше или равен правому операнду
Меньше или равно	<=	Возвращает true, если левый операнд меньше или равен правому операнду
Строгое равенство	===	Возвращает true, если оба операнда равны и относятся к одному типу
Строгое неравенство	!==	Возвращает true, если операнды не равны или не относятся к одному типу

Операции

Логические операции

- **позволяют сравнивать результаты работы двух условных операндов** с целью определения факта возвращения одним из них или обоими значения true и выбора соответствующего продолжения выполнения сценария.
- **можно применять при необходимости одновременной проверки более одного условия и использования результатов этой проверки.**

Операция	Символ	Функция
Логическое И	&&	Возвращает true, если операдны с обеих сторон от операции вернули значение true
Логическое ИЛИ	||	Возвращает true, если операнд с любой из сторон операции вернул true
Логическое НЕ	!	Операция логического НЕ является унарной. Действие операции ! заключается в том, что она меняет значение своего операнда на противоположное

Операции

Побитовые операции

Операция	Формат	Описание
Побитовое И	a & b	Ставит 1 для каждого бита результата, если соответствующие разряды у обоих операндов равны 1
Исключающее ИЛИ (XOR)	a ^ b	Ставит 1 для каждого бита результата, если только один из соответствующих битов операндов равен 1
Побитовое ИЛИ	a | b	Ставит 1 для каждого бита результата, если хотя бы один из соответствующих битов равен 1
Побитовое НЕ	~a	Это унарная операция, она заменяет каждый бит операнда на противоположный
Сдвиг влево	a << b	Сдвигает двоичное представление числа a на b разрядов влево, добавляя справа нули
Сдвиг вправо	a >> b	Сдвигает двоичное представление числа a на b разрядов вправо. Освобождающиеся разряды заполняются знаковым битом
Сдвиг вправо с заполнением нулями	a >>> b	Сдвигает двоичное представление числа a на b разрядов вправо, отбрасывая сдвигаемые биты и добавляя нули слева

Операторы

Условный оператор if

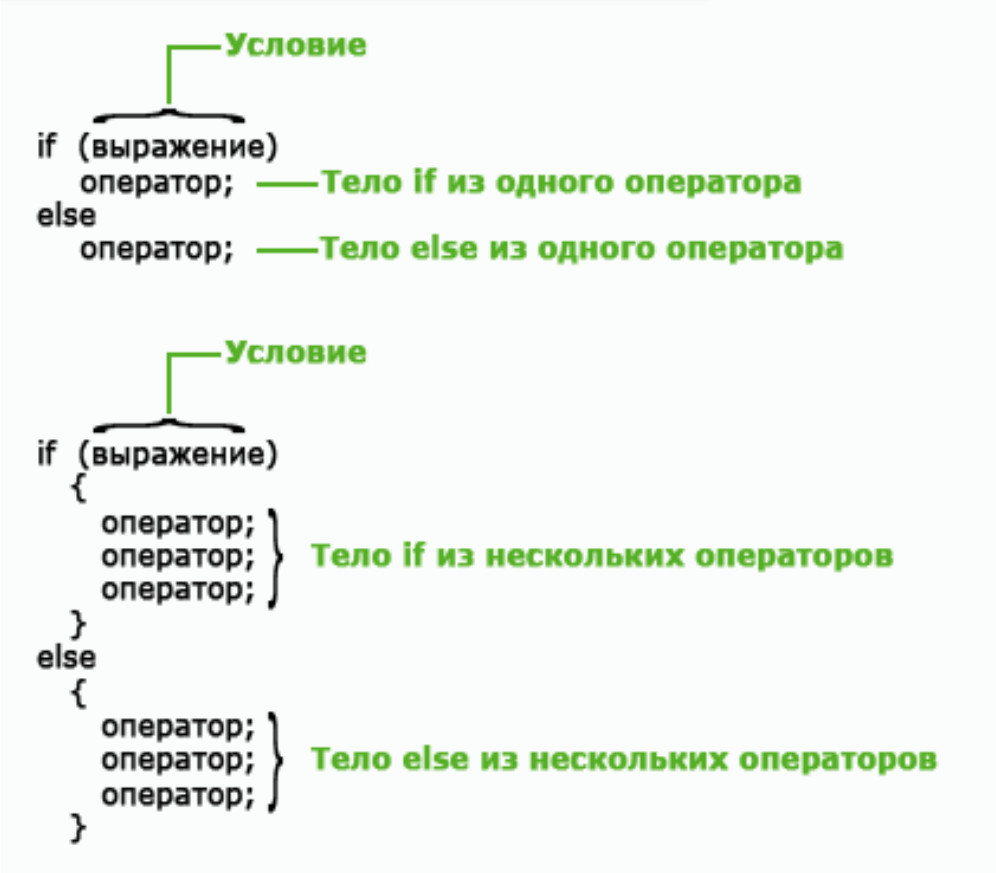

```
var num = prompt("Введите любое число","");
if(num > 5) {
  document.write("Число: ", num, " больше 5");
}
```

Оператор не обязательно записывать именно под if, если запись не большая, то ее можно написать и **в одну строку**:

```
if(num > 5) x = y;
```

Операторы

Оператор if … else




```
<html>
<body>
 <script>
  var num = prompt("Введите
любое число","");

 if(num >= 10){
    document.write("Число: ", num, "
больше или равно 10");
  }

 else {
    document.write("Число: ", num, "
меньше 10");
  }
</script>
</body>
</html>
```

Операторы

Вложенный оператор if

```
if(условие)
{
  if(условие) оператор 1;
  if(условие) оператор 2;
  else оператор 3; //этот else ассоциирован с if(b)
}
else оператор 4; //этот else ассоциирован с if(i)
```

Конструкция if-else-if

```
if(условие){
  оператор;
} else if(условие){
  оператор;
} else if(условие){
  оператор;
}
else
  оператор;
```

Операторы

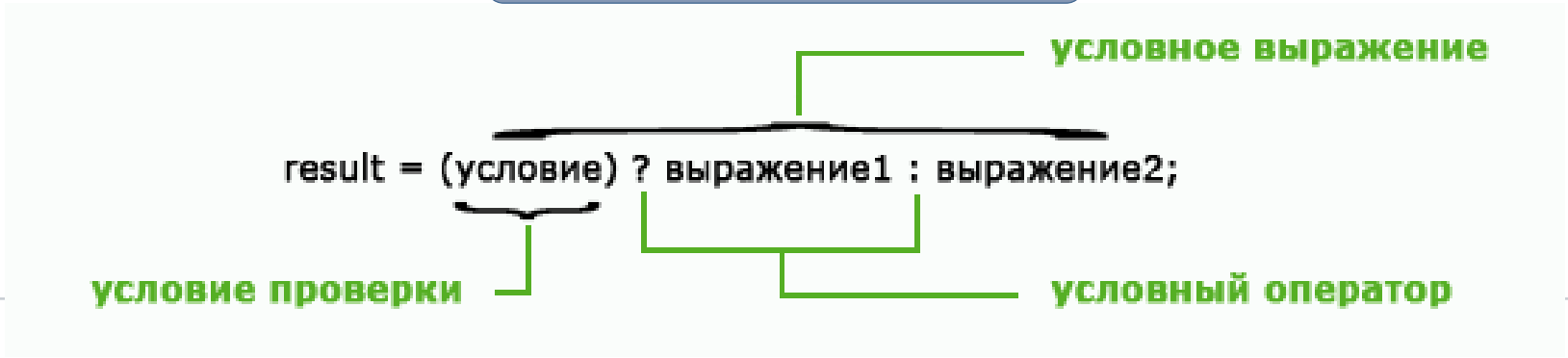
Условный оператор

В данном примере переменной *x* присваивается наименьшее из значений *a* и *b* с помощью конструкции if...else:

```
if (a < b)
  x = a;
else
  x = b;
```

С помощью условного оператора предыдущий код можно записать следующим образом:

```
x = (a < b) ? a : b;
```



Операторы

Оператор switch

Сравнивает значение переменной с различными вариантами. При сравнении используется операция строгого равенства "===".

```
var x = 3;
switch(x){
case 1:                          //if(x === 1)
    document.write("x равен 1");
    break;
case 2:                          //if(x === 2)
    document.write("x равен 2");
    break;
case 3:                          //if(x === 3)
    document.write("x равен 3");
    break;
}
```

Операторы

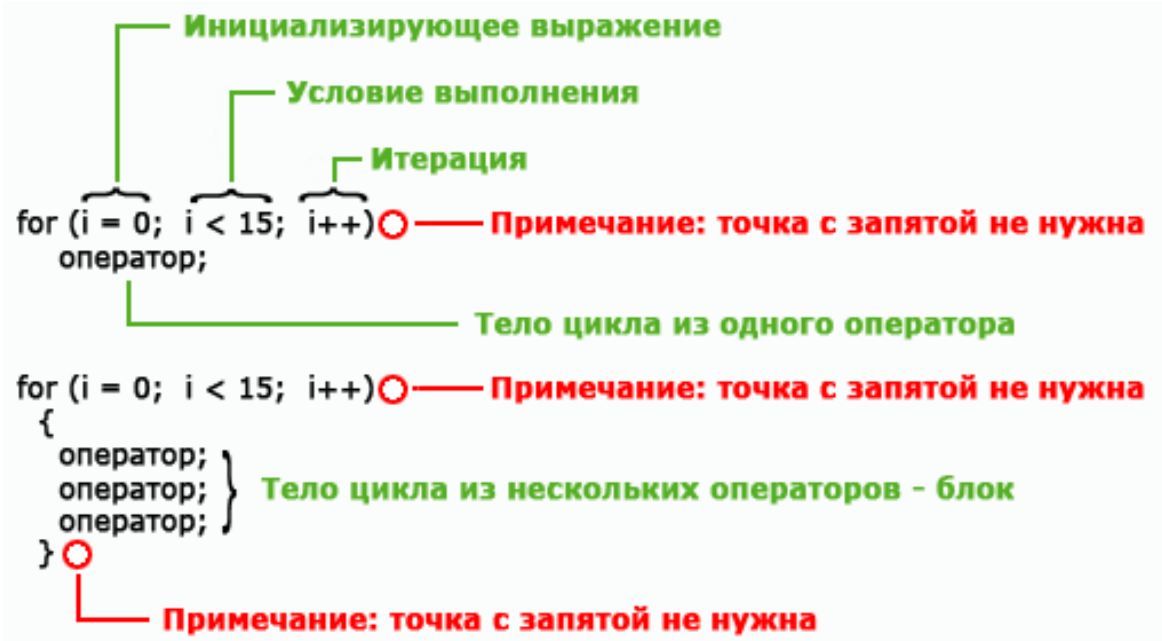
Ключевое слово default

Предназначено для того, чтобы программа могла **выполнить** некоторую последовательность действий **в том случае, если ни одно из значений констант не совпало со значением переменной в операторе switch.**

```
var x = 3+3;
switch(x){
case 1:
    document.write("x равен 1");
    break;
case 2:
    document.write("x равен 2");
    break;
default:
    document.write("С такими значениями не работаю");
}
```

Циклы

Цикл for



Пример (на экран выводятся квадраты целых чисел от 0 до 14):

```
var i;
for(i = 0; i < 15; i++){
  document.write("квадрат числа " +
i + " = " + (i * i) + "<br>");
}
```

Любая часть цикла может отсутствовать:

```
for(var x = 0; x < 5;){
  document.write(x + " ");
  x++;      //итерация находится в теле цикла
}

//можно вообще все убрать, получив
бесконечный цикл
for(;;){
//...
                    }
```

Циклы

Цикл while

Содержит условие выполнения цикла, но не содержит ни инициализирующих, ни инкрементирующих выражений:

```
while(условие){
  //блок кода
}
```

```
var n = 0;
while(n!= 5){
  document.write(n + " ");
  n++;  //если из кода убрать эту строку, то цикл будет бесконечным
}
```

Циклы

Цикл do … while

Когда необходимо **выполнить тело цикла хотя бы один раз вне зависимости от истинности проверяемого условия**, следует использовать цикл **do … while,** в котором **условие выполнения цикла** располагается **не перед, а после тела цикла:**

```
do
  {
  //блок кода
}
while (условие);
```

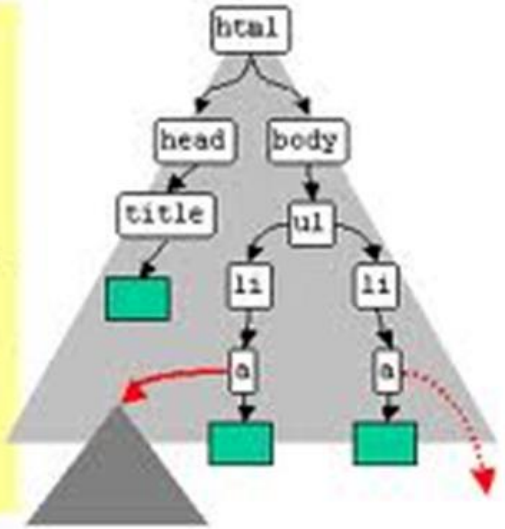
```
do
  {
  x=x + "The number is " + i + "<br>";
  i++;
  }
while (i<5);
```

DOM – объектная модель документа

Управление WEB контентом происходит через **объектную модель документа** (**DOM** - Document Object Model).

DOM W3C

```
<html>
 <head>
  <title>Portals</title>
 </head>
 <body>
  <ul>
   <li>
    <a href="_">Yahoo</a>
   </li>
   <li>
    <a href="_">Lycos</a>
   </li>
  </ul>
 </body>
</html>
```



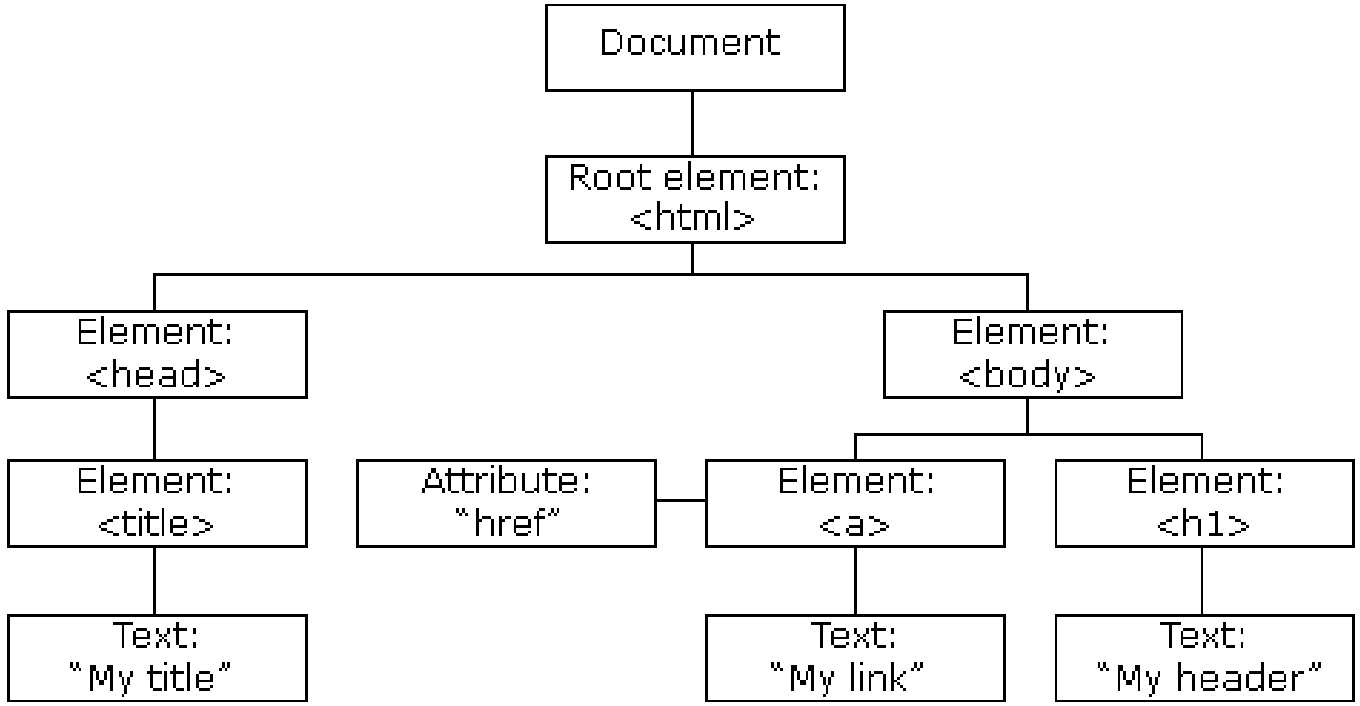
DOM является стандартом, предложенным веб-консорциумом, и регламентирует способ представления содержимого документа (в частности веб-страницы) в виде набора объектов.

Под содержимым понимается все, что может находиться на веб-странице: рисунки, ссылки, абзацы, текст и т. д.

DOM – объектная модель документа

Когда веб-страница загружена, браузер создает **объектную модель документа (DOM - Document Object Model)** этой страницы.

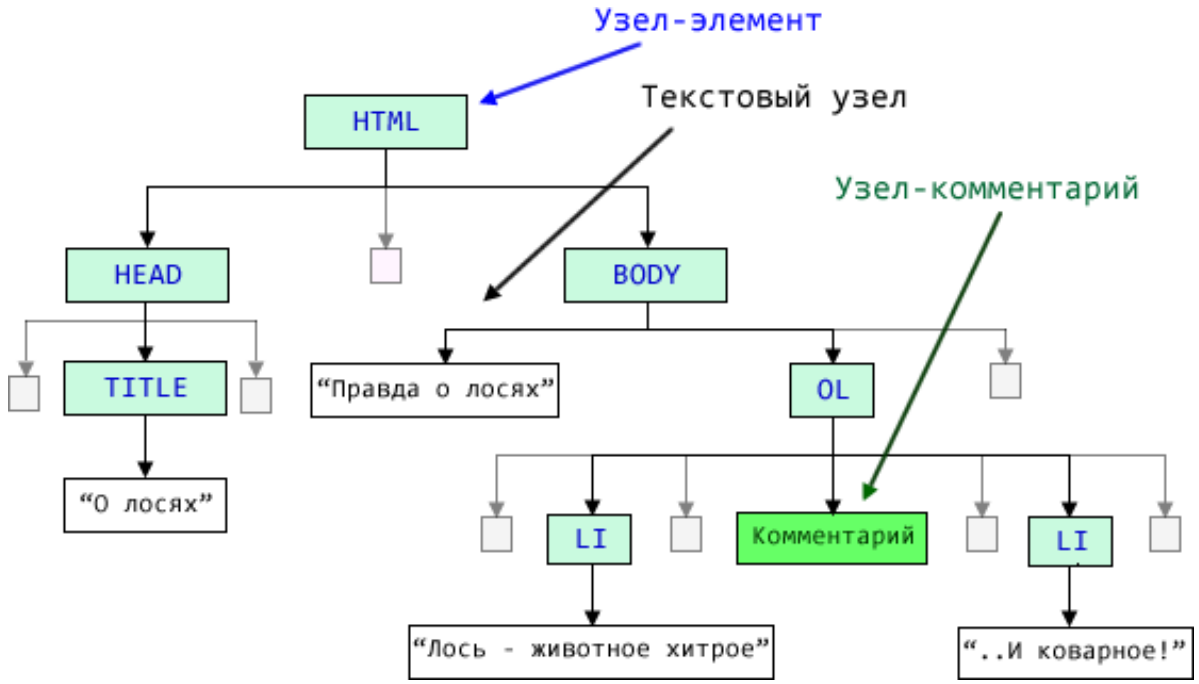
DOM-модель HTML-страницы строится как **дерево объектов**:



DOM – объектная модель документа

Теги образуют **узлы-элементы** (element node). Текст представлен **текстовыми узлами** (text node). И то и другое - равноправные узлы дерева DOM.

```
<html>
    <head>
        <title>
            О лосях
        </title>
    </head>
    <body>
        Правда о лосях.
        <ol>
            <li>
            Лось - животное хитрое
            </li>
            <li>
            .. И коварное
            </li>
        </ol>
    </body>
</html>
```



DOM – объектная модель документа

Зачем нужна модель DOM?

DOM нужен для того, чтобы манипулировать страницей — читать информацию из HTML, создавать и изменять элементы.

Модель DOM абстрагирует содержание от грамматики.

Модель DOM гарантирует правильную грамматику и правильное оформление документов.

Модель DOM напоминает структуры иерархических и реляционных баз данных.

DOM – объектная модель документа

С помощью JavaScript и с использованием DOM, на HTML странице можно выполнять следующие манипуляции:

- **изменять HTML-элементы;**
- **изменять CSS-стили;**
- **изменять атрибуты HTML-элементов;**
- **реагировать на все события.**

Стратегии поиска HTML-элементов

Стандарт DOM предусматривает несколько средств поиска элемента. Это методы:

- **getElementById**
- **getElementsByTagName**
- **getElementsByName**

Частая опечатка связана с отсутствием буквы **s** в названии метода getElementById, в то время как в других методах эта буква есть: getElement**s**ByName.

Стратегии поиска HTML-элементов

Поиск по id

Наиболее удобный и простой способ.
Используется вызов: **document.getElementById(id)**

```
document.getElementById('dataKeeper').style.color = 'blue';
/*изменение цвета текста на голубой элемента с id="dataKeeper"*/
```

```
var x=document.getElementById("intro");
/*поиск элемента с id="intro" */
```

- Если искомого элемента не существует, то getElementById возвращает null.
- Если есть много элементов с таким ID (что само по себе неправильно), то, в зависимости от браузера, getElementById может возвращать совершенно разные результаты. Например, первый элемент с этим ID.

Стратегии поиска HTML-элементов

```
<html>
<head>
<script>
function changeLink()
{
document.getElementById('myAnchor').innerHTML="Yandex";
document.getElementById('myAnchor').href="http://www.yandex.ru";
document.getElementById('myAnchor').target="_blank";
}
</script>
</head>
<body>
<a id="myAnchor" href="http://www.microsoft.com">Microsoft</a>
<input type="button" onclick="changeLink()" value="Change link">
</body>
</html>
```

Стратегии поиска HTML-элементов

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Web Page</h1>
<p id="demo">My First Paragraph.</p>
    <script>
    document.getElementById("demo").innerHTML="My First JavaScript";
    </script>
</body>
</html>
```

Стратегии поиска HTML-элементов

Поиск по Tag Name

Используется, чтобы получить все элементы с определенным тегом, и среди них искать нужный.

document.getElementsByTagName(tag).

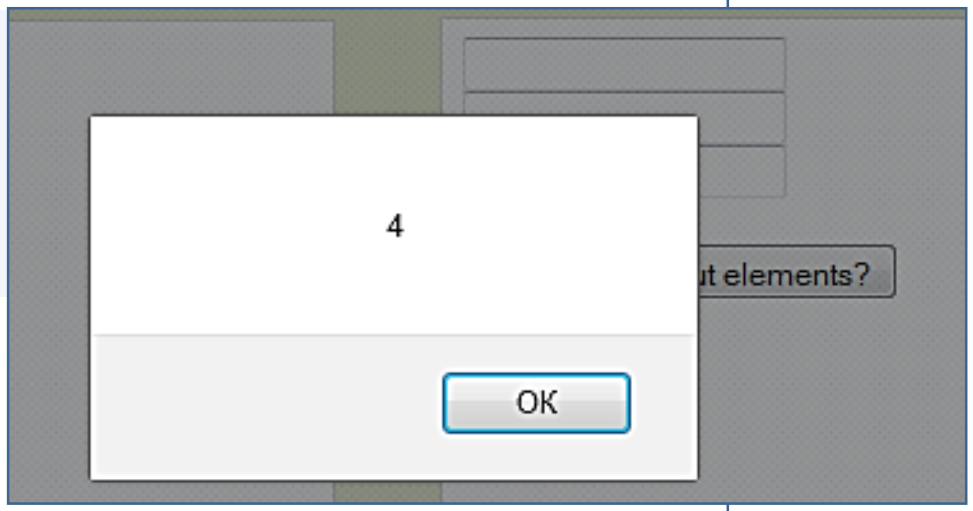
Возвращает массив из элементов, имеющих такой тег.

```
var x=document.getElementById("main");
var y=x.getElementsByTagName("p");
/*поиск элементов с id="main", и после этого поиск всех <p> элементов внутри "main" */
```

```
document.getElementsByTagName("DIV")[0].getElementsByTagName("LI");
/*поиск списка элементов LI, находящихся внутри первого тега div */
```

Стратегии поиска HTML-элементов

```
<html>
<head>
<script>
function getElements()
  {
  var x=document.getElementsByTagName("input");
  alert(x.length);
  }
</script>
</head>
<body>
<input type="text" size="20"><br>
<input type="text" size="20"><br>
<input type="text" size="20"><br><br>
<input type="button" onclick="getElements()" value="How many input elements?">
</body>
</html>
```



Стратегии поиска HTML-элементов

Поиск по Name

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

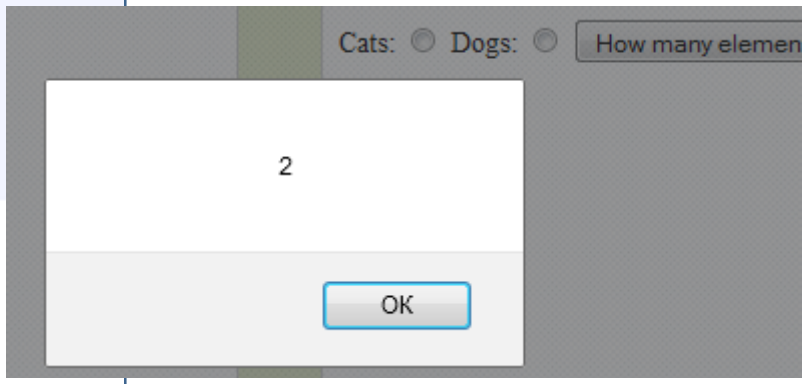
document.getElementsByName(name).

Он работает только с теми элементами, для которых в спецификации явно предусмотрен атрибут name: это **form**, **input**, **a**, **select**, **textarea** и ряд других, более редких.

```
var x=document.getElementsByName("x");
alert(x.length);
/* вывод количества элементов с заданным именем "x" */
```

Стратегии поиска HTML-элементов

```
<!DOCTYPE html>
<html>
<head>
<script>
function getElements()
{
var x=document.getElementsByName("x");
alert(x.length);
}
</script>
</head>
<body>
Cats:
<input name="x" type="radio" value="Cats">
Dogs:
<input name="x" type="radio" value="Dogs">
<input type="button" onclick="getElements()"
       value="How many elements named 'x'?">
</body>
</html>
```



Валидация данных

JavaScript можно использовать для проверки данных в HTML формах до передачи контента на сервер.

- **Пользователь оставил пустыми обязательные поля?**
- **Пользователь ввел некорректный e-mail адрес?**
- **Пользователь ввел некорректную дату?**
- **Пользователь ввел текст в числовые поля или наоборот?**

Валидация данных

```
<!DOCTYPE html>
<html>
<head>
<script>
function validateForm()
{
var x=document.forms["myForm"]["fname"].value;
if (x==null || x=="")
  {
  alert("First name must be filled out");
  return false;
  }
}
</script>
</head>
<body>
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm()" method="post">
First name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

Материалы для самостоятельного изучения

- http://www.w3schools.com/
- https://html5book.ru/
- http://htmlbook.ru/
- https://www.codecademy.com
- https://htmlacademy.ru/
- https://www.w3schools.com/js/js_best_practices.asp

Q&A

Thank You

Focused forward

Netcracker