# Intro

## Intro

### Script tag

<!--Тег script-->

<script type="text/javascript">

alert("Hello world!");

document.write("Hello world again!");

</script>

### Comments

<!--Тег script-->

<script type="text/javascript">

// Вы можете оставлять как однострочный комментарий

alert("Hello world!");

/\*

Так и многострочный

комментарий

\*/

</script>

<!--Завершение тега script-->

### Variables

<script type="text/javascript">

// Создаем переменную с именем - 'а' и присваиваем ей значение - 10.

var a = 10;

var b;

var d;

//String s = "hello"; // недопустимо явно указывать тип переменной.

b = "Some string";

// Возможно создание переменной без использования ключевого слова - var

e = 22;

alert(e);

var a = b;

alert(a);

</script>

### Именование переменных

Ограничения которые следует учитывать при именовании переменных:

1. Нельзя в имени переменной использовать символ - пробел.

alice variable - не правильно

alice\_variable - правильно

2. Не начинайте имя переменной с числа

1variable - не правильно

varia23ble1 - правильно

3. Переменная может начинаться с символа $

$variable

4. При именовании часто используются правила camelCase или PascalCase

-->

<html>

<head>

<title>Именование переменных</title>

<script type="text/javascript">

var a = "Some string with spaces";

//alert(A); // не сработает так как переменной "A" не существует.

alert(a);

</script>

## DataType

### Number

Все числа хранятся в формате float64 - 8 байт с плавающей точкой.

В этом формате не всегда возможны точные вычисления.

Infinity

NaN - не число

Number.MAX\_VALUE - максимально допустимое значение в JavaScript +9007199254740992

Number.MIN\_VALUE - минимально допустимое значение в JavaScript -9007199254740992

Number.NaN - не число

Number.POSITIVE\_INFINITY - + бессконечность 1.7976931348623157E+10308

Number.NEGATIVE\_INFINITY - - бессконечность -1.7976931348623157E+10308

-->

<html>

<head>

<title>Интеграция JavaScript</title>

<script type="text/javascript">

var a = 10;

var b = 0.8;

var c = a + b;

document.write("a + b = " + c + "<br/><hr/>");

document.write("Number.NaN = " + Number.NaN + "<br/>");

document.write("Number.MAX\_VALUE = " + Number.MAX\_VALUE + "<br/>");

document.write("Number.MIN\_VALUE = " + Number.MIN\_VALUE + "<br/>");

document.write("Number.POSITIVE\_INFINITY = " + Number.POSITIVE\_INFINITY + "<br/>");

document.write("Number.NEGATIVE\_INFINITY = " + Number.NEGATIVE\_INFINITY + "<br/>");

</script>

### String

Строки - тип данных для представления текста (в строку может входить любой UNICODE символ)

строковые литералы нужно заключать в согласованные пары одинарных или двойных кавычек.

Конкатенация (лат. concatenatio «присоединение цепями; сцепле́ние») —

операция склеивания объектов линейной структуры, обычно строк.

-->

<html>

<head>

<title>Строки, конкатенация</title>

<script type="text/javascript">

var string1 = "Hello ";

var string2 = 'World';

var resString = string1 + string2; // конкатенация значений переменных string1 и string2

document.write(string1 + "<br/>");

document.write(string2 + "<br/>");

document.write(resString + "<br/><hr>");

var str = "10";

var a = 1;

// Конкатенация

// Произошло неявное преобразование значения типа Number в тип String.

// Значение в переменной (a) неявно преобразуется в строку (т.е в значение типа String)

// и далее над этой строкой-числом производится операция конкатенации.

document.write(str + a + "<br/>");

// Произошло неявное преобразование значения типа String в тип Number.

// Значение в переменной str неявно преобразуется в число (т.е в значение типа Number)

// и далее над этим числом производится арифметическая операция.

document.write(str - a + "<br/>");

document.write(str \* "2" + "<br/>");

document.write(str / "2" + "<br/>");

</script>

### Logic

Логический тип данных.

Данные хранящиеся в переменной булевого типа могут принимать два значения:

истина (true) и ложь (false).

Часто истина представляется единицей, а ложь — нулем.

-->

<html>

<head>

<title>Строки, конкатенация</title>

<script type="text/javascript">

var bool1 = true;

var bool2 = false;

document.write(bool1+"<br/>");

document.write(bool2 + "<br/><hr/>");

bool1 = 1;

bool2 = 0;

document.write(bool1 + "<br/>");

document.write(bool2 + "<br/>");

</script>

</head>

### Null

Null - это тип данных, указывающий на отсутствие объекта.

В JavaScript null не является «ссылкой на несуществующий объект»

или «нулевым указателем», как в некоторых других языках.

Null - значение, аналогичное понятиям: «ничего» или «значение неизвестно».

-->

<html>

<head>

<title>Null</title>

<script type="text/javascript">

var a = null;

alert(a);

alert(b); // Ошибка!

</script>

### Undefined

undefined - это тип данных, который используется для переменных или свойств объекта,

значения для которых не существует или оно не присвоено.

Единственным значением неопределенного типа, которое может иметь переменная,

является значение undefined.

-->

<html>

<head>

<title>Undefined</title>

<script type="text/javascript">

var a;

alert(a); // undefined

alert(b); // Ошибка!

</script>

## Функции

### Document.write

document.write(...) — метод, выводящий на страницу переданные ему аргументы (параметры).

Аргументов может быть любое количество, и они могут быть любых типов,

при выводе значения аргументов преобразуются в строки.

Функция document.writeln(), аналогична document.write(),

но добавляет в конце своего вывода перевод строки.

(Не во всех браузерах работает корректно.)

-->

<html>

<head>

<title>Интеграция JavaScript</title>

<script type="text/javascript">

var a = 10;

var b = "Some String";

var c = true;

document.write("Hello");

document.write("<br />"); // В качестве аргументов можно передавать любую HTML разметку.

document.writeln(a);

document.writeln(b);

document.writeln(c);

document.write("<hr />");

</script>

### Prompt

Метод prompt(1, 2) - принимает два аргумента:

1 - выводится в качестве простой строки в модальном окне.

2 - значение по умолчанию в текстовом поле для ввода.

Метод prompt(1, 2) - выводит сообщение в окне с текстовым полем и двумя кнопками: "ОК" и "ОТМЕНА".

Метод prompt(1, 2) - возвращает введенное значение или null, если пользователь нажал на кнопку "ОТМЕНА".

Как и в alert, prompt - модальное окно

(пользователь не может продолжить работу, пока не нажмет на одну из кнопок).

-->

<html>

<head>

<title>Метод prompt()</title>

<script type="text/javascript">

var a = prompt("Введите какое либо значение", "сюда");

if (a != null)

alert(a);

else { alert("Blya");}

</script>

### Confirm

confirm() - [подтвердить], действует похоже на alert, за исключением того,

что добавляет кнопку "Отмена" в диалоговое окно.

Возвращает true, если нажата кнопка - OK или false , если нажата кнопка - Cancel.

-->

<html>

<head>

<title>Метод prompt()</title>

<script type="text/javascript">

var conf = confirm("Вы уверены?");

document.write(conf);

</script>

### ParseInt ParseFloat

<script type="text/javascript">

var x = prompt("Input x value ", "0"); // х - будет строкового типа

var y = prompt("Input y value ", "0"); // y - будет строкового типа

var result = x + y;

document.write(result + "<hr/>");

// Конвертация строкового значения в числовое:

// parseFloat() конвертация в вещественное значение.

// parseInt() конвертация в целочисленное значение.

// Данные функции возвращают значение NaN если конвертация не удалась.

var integerX = parseInt(x);

var integerY = parseInt(y);

result = integerX + integerY;

document.write(result);

</script>

## Operators

### Операторы преобразования

var a = -8;

var string = "10";

// Унарный минус (-)

result = -a; // Операция смены знака на противоположный.

document.write("-a = " + result + "<br />");

// Унарный плюс (+)

// не изменяет знака при применении к числовому типу (ничего не происходит).

result = +a;

document.write("+a = " + result + "<br />");

// Унарный плюс (+)

// при применении к строковому типу, преобразовывает его в числовой.

result = +string + a;

document.write("+string + a = " + result + "<br />");

Increment

var a = 9;

document.write("Постфиксный инкремент <br />");

document.write(a + "<br />"); // выведет 9

a++;

document.write(a + "<br />"); // выведет 10

document.write(a++ + "<br />"); // выведет 10

document.write(a + "<br />"); // выведет 11

document.write("<hr/>");

document.write("Префиксный инкремент <br />");

document.write(a + "<br />"); // выведет 11

++a;

document.write(a + "<br />"); // выведет 12

document.write(++a + "<br />"); // выведет 13

document.write(a + "<br />"); // выведет 13

### Equality

Оператор равенства "==" предназначен для сравнения значений двух переменных,

при этом необходимо помнить о следующих правилах:

1. если одно значение null а другое undefined - значения равны.

2. если одно значение число, а второе строка, то строка преобразоваться в число и производиться сравнение.

3. если одно значение true оно преобразуется в 1, если значение false оно преобразоваться в 0.

-->

<html>

<head>

<title>Операторы равенствам</title>

<script type="text/javascript">

var a = 1; // простое число

var b = "1"; // число как строка

var c = "some"; // строка

var d = 'som'; // строка

var e = true; // логическое "истина"

var res;

res = a == b;

document.write("a равно b : " + res + "<br />");

res = c == d;

document.write("c равно d : " + res + "<br />");

res = a == e;

document.write("a равно e : " + res + "<br />");

</script>

### Identification

Идентичность - требует совпадения типа и значения переменной,

когда для равенства достаточно только совпадающих значений.

Оператор идентичности "===" предназначен для проверки значений двух переменных на идентичность,

при этом необходимо помнить о следующих правилах:

1. если значения имеют разные типы они не идентичны.

2. если значения являются числами имеют одинаковые значения и не являются значения NaN - они идентичны.

3. если две строки имеют одинаковую длину и содержат одни и те же символы - они идентичны.

4. два одинаковых логических значения идентичны.

-->

<html>

<head>

<title>Операторы равенства</title>

<script type="text/javascript">

var a = 1; // простое число

var b = "1"; // число как строка

var c = "some"; // строка

var d = 'some'; // строка

var e = true; // логическое "истина"

var f = 1;

var g = true;

var res;

res = a == b;

document.write("a равно b : " + res + "<br />");

res = a === b;

document.write("a идентично b :" + res + "<br />");

document.write("<hr/>");

res = a == f;

document.write("a равно f :" + res + "<br />");

res = a === f;

document.write("a идентично f :" + res + "<br />");

document.write("<hr/>");

res = c == d;

document.write("c равно d :" + res + "<br />");

res = c === d;

document.write("c идентично d :" + res + "<br />");

document.write("<hr/>");

res = f == g;

document.write("f равно g :" + res + "<br />");

res = f === g;

document.write("f идентично g :" + res + "<br />");

### NOT

Оператор неравенства "!=" возвращает противоположное значение оператора равенства.

Оператор неидентичности "!==" возвращает противоположное значение оператора идентичности.

-->

<html>

<head>

<title>Операторы равенства</title>

<script type="text/javascript">

var a = 1; // простое число

var b = "1"; // число как строка

var c = "some"; // строка

var d = 'same'; // строка

var e = true; // логическое "истина"

var res;

res = a != b;

document.write("a не равно b : " + res + "<br />");

res = c !== d;

document.write("c не идентично d : " + res + "<br />");

res = a !== e;

document.write("a не идентично e : " + res + "<br />");

### String equation

Сравнение строк производиться посимвольно для числовых значений каждого символа из кодировки Unicode.

Символ в верхнем регистре считается меньше символа в нижнем регистре.

Например: A - code (41 Hex)

a - code (61 Hex)

-->

<html>

<head>

<title>Операторы</title>

<script type="text/javascript">

res = "A" < "a";

document.write("A < a = " + res + "<br /><hr/>");

var c = "hello";

var d = "hallo";

res = c < d;

document.write(rez);

# Logical Struction

## Conditional statement

### If else inner

var a = prompt("Введите слово Hello, или World", "Hello");

if (a == "Hello" || a == 'hello') {

document.write("Введенное значение верно и равно Hello");

}

else if (a == "World" || a == 'world') { // Выполнится если выражение в первой конструкции будет false

document.write("Введенное значение верно и равно World");

}

else {

document.write("Введенное значение НЕ верно");

}

document.write("<br/>Эта строка выполнится в любом случае");

### Switch

// Предлагаем пользователю ввести число и принимаем ввод от пользователя.

var number = prompt("Введите число 1 или 2:");

// Создаем конструкцию многозначного выбора, переключатель - switch,

// которому в качестве выражения-селектора, передаем переменную - day.

// ВНИМАНИЕ!

// Тип значения выражения селектора и значений постоянных выражений операторов case должны совпадать.

switch (number) { // (number) — выражение-селектор.

// В теле переключателя switch-case, создаем несколько операторов case,

// c постоянными выражениями равными "1" и "2" соответственно.

// Если значение выражения-селектора совпадет с одним из значений постоянных выражений,

// то выполняется тело оператора case, постоянное выражение которого, совпало с выражением-селектора.

case "1": // "1" — постоянное выражение.

{

document.write("Один"); // Ветвь 1

};

break; // Выход из переключателя.

case "2": // "2" — постоянное выражение.

{

document.write("Два"); // Ветвь 2

};

break; // Выход из переключателя.

// Создаем (необязательный) оператор default, тело которого выполняется в случае,

// если значение выражения-селектора не совпадет ни с одним из значений постоянных выражений.

default:

{

document.write("Вы ввели значение отличное от 1 и 2."); // Ветвь 8

};

// Если в переключателе отсутствует оператор default, то управление передается за пределы переключателя switch.

### Switch 2

var day = prompt("Введите номер дня недели: 1-2-3-4-5-6-7: ", "");

switch (day) {

case "1":

case "2":

case "3":

case "4":

case "5":

document.write("Этот день недели - Рабочий.");

break;

case "6":

case "7":

{

document.write("Этот день недели - Выходной.");

}

break;

default:

{

document.write("Вы ввели несуществующий день недели.");

}

break;

}

### Ternar

Тернарная условная операция.

Тернарный (троичный) оператор [?:] .

Тернарный оператор [?:], является сокращенной формой конструкции if ... else.

[?:] - включает в себя три операнда - ( условие ? Если условие ИСТИННО : Если условие ЛОЖНО )

Алгоритм работы тернарной операции: (логическое выражение) ? выражение 1 : выражение 2

1. Вычисляется логическое выражение (условие).

2. Если логическое выражение истинно, то вычисляется значение выражения выражение 1,

в противном случае — значение выражения выражение 2.

3. Вычисленное значение возвращается.

-->

<html>

<head>

<title>Тернарный оператор</title>

<script type="text/javascript">

var a = 1, b = 2, max = 0;

// Переменной max, присваиваем возвращаемое значение тернарного оператора.

max = (a > b) ? a : b; // ЕСЛИ: (a > b) ТО: вернуть a ИНАЧЕ: вернуть b;

document.write(max);

</script>

## Cyclic statement

### Factorial

var counter = prompt("Введите число");

var factorial = 1;

document.write("Факториал числа: " + counter + "! = ");

do {

if (counter == 0) {

factorial = 1;

break;

}

factorial = factorial \* counter;

counter = counter - 1;

}

while (counter > 0);

document.write(factorial);

## Label

Label: for (var j = 0; j < 2; j++) {

for (var i = 0; i < 2; i++) {

var input = prompt("Введите любой текст, чтобы выйти нажмите отмена", '');

if (input == null)

break Label; //выйти сразу из 2-х циклов.

//continue Label;

document.write(input + " ");

}

document.write("<br/>");

}

document.write('Готово!');

# Arrays

## Arrays

### Array init

Массив - обьект, проиндексированый набор определенных элементов.

В отличии от прочих языков программирования в JavaScript, не обязательно элементами

массива должны быть однотипные значения.

При создании массива используются [] скобки, внутри которых через заяпятую перечисляются элементы.

Каждый элемент нумеруется индексом по возрастанию от 0.

-->

<html>

<head>

<title>Массивы в JavaScript</title>

<script type="text/javascript">

// Одномерный массив из целых чисел размерностью в 5 элементов.

var a = [1, 2, 3, 4, 6];

document.write(a); // Вывод элементов массива, при этом массив преобразуется в строку "1, 2, 3, 4, 6"

// Одномерный массив из строковых элементов размерностью в 12 элементов.

var b = ["September", "October", "November", "December", "January", "Fabruary", "March", "April", "May", "June", "Jull", "August"];

document.write("<hr />");

document.write(b);

// Пустой массив размерностью в 3 элемента.

var c = [, , , ];

document.write("<hr />");

document.write(c);

// Одномерный массив из трех элементов разных типов.

var d = ["Hello", 34, true];

document.write("<hr />");

document.write(d);

### Array constructor

Массив является обьектом, следовательно для него в JavaScript предусмотрен специальный конструктор Array(),

в зависимости от аргументов он строит массив:

1. Конструктор без аргументов - создает пустой массив.

2. Конструктор с одним целочисленным аргументом - создаст массив размерностью в количество элементов указанное в аргументе.

3. Конструктор с двумя и более аргументами - создает массив из передаваемых аргументов,

при этом все аргументы будут расположены в массиве так как заданы в аргументах.

-->

<html>

<head>

<title>Конструктор Array()</title>

<script type="text/javascript">

var a = Array(); // Пустой массив.

var b = Array(1, 2, 3, 4, 5, 6); // Массив размерностью в 6 элементов.

document.write(b);

document.write("<hr />");

var c = Array(10); // Массив размерностью в 10 элементов.

document.write(c);

### Array index init

Обращаться к элементам массива можно по индексу.

Для этого используеться форма: имя\_массива[индекс массива].

При таком обращении возможна как запись так и чтение элемента.

-->

<html>

<head>

<title>Чтение и запись в массив</title>

<script type="text/javascript">

var arr = Array(5); // Создаем массив размерностью в 5 элеемнтов.

arr[0] = 2; // Элементу с индексом 0 присваиваем значение 2.

arr[1] = 5;

arr[2] = 12;

arr[3] = 8;

arr[4] = 23;

document.write("Содержимое массива" + arr);

arr[4] = "Четыре"; // Изменяем значение элемента массива по индексу 4.

arr[10] = "Десять"; // Происходит изменение размера массива.

document.write("<hr />");

document.write(arr);

### Array length

// Длина массива = 5 элементов.

var array = ['h', 'e', 'l', 'l', 'o'];

document.write("Длинна массива [" + array + "] равна =" + array.length);

### Array resize

// Длина массива = 5 элементов.

var array = ['h', 'e', 'l', 'l', 'o'];

// Массив увеличил свой размер на два элемента, новая длина - 7 элементов.

array.length = 7;

for (var i = 0; i < array.length; i++) {

document.write(array[i] + "<br />");

}

document.write("<hr />");

// Значения всех элементов, которые не попали в новую заданную длину массива, будут утеряны.

array.length = 2;

for (var i = 0; i < array.length; i++) {

document.write(array[i] + "<br />");

}

### Array resize by index

Свойство length открыто как для чтения так и для записи,

при этом возможно и увеличение длины так и уменьшение длины массива.

-->

<html>

<head>

<title>Длина массива</title>

<script type="text/javascript">

var arr = [111, 222, 333];

document.write("Длина массива " + arr.length + "<br />");

// При присвоении значения по индексу, которого нет в массиве, происходит добавление нового элемента массива.

arr[10] = 777;

document.write("Новая длина массива " + arr.length + "<br/><hr/>");

// Значение свойства arr.length - всегда на единицу больше индекса последнего элемента в массиве.

for (var i = 0; i < arr.length; i++) {

document.write(arr[i] + "<br />");

}

### Array associate

Ассоциативный массив - абстрактный тип данных, позволяющий хранить пары вида «(ключ, значение)».

От простых массивов отличается тем, что доступ к элементам происходит по ключу.

-->

<html>

<head>

<title>Аcсоциативные массивы</title>

<script type="text/javascript">

// Объект, имеющий три свойства.

var laptop = { cpu: "Core i7", ram: "4 GB", screen: "15" };

// Запрос на введение имени свойства.

var key = prompt("Введите интересующий параметр (ram, cpu, screen)", "");

// Обращение к массиву по ключу.

var value = laptop[key];

document.write(value + "<hr/>");

// Обращение к массиву по ключу.

// Данные выражения имеют одинаковое значение:

document.write("<p>" + laptop.cpu);

document.write("<p>" + laptop["cpu"]);

### Two dimensional array

// Создание многомерного массива.

var table = new Array(10);

for (var i = 0; i < table.length; i++) { // В таблице 10 строк

table[i] = new Array(10); // В каждой строке 10 столбцов

}

// Инициализация массива.

for (var row = 0; row < table.length; row++) {

for (var col = 0; col < table[row].length; col++) {

if ((row + col) % 2 == 1)

table[row][col] = "<img src='images/parquet1.jpg' width='50' height='50'/>";

else

table[row][col] = "<img src='images/parquet2.jpg' width='50' height='50'/>";

}

}

table[8][1] = "<img src='images/parquet3.jpg' width='50' height='50'/>";

for (var row = 0; row < table.length; row++) {

for (var col = 0; col < table[row].length; col++) {

document.write(table[row][col]);

}

document.write("<br/>");

}

var prod = table[8][1];

document.write("<hr/>" + prod);

## Array methods

### Join

Метод Join() - преобразует все элементы массива в строки и объединяет их.

Если метод принимает строковой аргумент, то аргумент станет разделителем вместо запятых.

-->

<html>

<head>

<title>Метод Join()</title>

<script type="text/javascript">

var arr = ["September", "October", "November", "December", "January", "Fabruary", "March", "April", "May", "June", "Jull", "August"];

document.write(arr);

document.write("<p><b>Значение которое возвращает метод - join(' | ');</b>");

var res = arr.join(" | ");

document.write("<p>" + res);

document.write("<p><b>Значение которое возвращает метод - join(< br/ >);</b>");

res = arr.join("<br/>");

document.write("<p>" + res);

### Join example

var arr = ["September", "October", "November", "December", "January", "Fabruary", "March", "April", "May", "June", "Jull", "August"];

document.write("<ul> <li>");

var rez = arr.join("</li><li>");

document.write(rez);

document.write("</li></ul>");

### Concat

concat() - дополняет элементы массива указанными элементами в параметрах.

-->

<html>

<head>

<title>Метод concat()</title>

<script type="text/javascript">

var a = [1, 2, 3];

a = a.concat(4, 5);

document.write("<p>" + a); // 1,2,3,4,5

a = a.concat(6, 7, [8, 9]);

document.write("<p>" + a); // 1,2,3,4,5,6,7,8,9

### Reverse

var a = new Array(1, 2, 3, 4, 5, 6, 7);

a.reverse(); // Меняет порядок следования элементов массива на противоположный.

document.write(a.join("<br/>"));

### Slice

slice() - Возвращает фрагмент или подмассив указанного массива.

-->

<html>

<head>

<title>Метод slice()</title>

<script type="text/javascript">

var a = [1, 2, 3, 4, 5, 6, 7];

// slice(начальный индекс (включительно), конечный индекс (не включительно));

var temp = a.slice(0, 3); // возвращает первые три элемента 1, 2, 3

document.write("<p>" + temp);

// slice(начальный индекс); - возвращает подмассив от указанного индекса до конца массива.

temp = a.slice(3); // 4, 5, 6, 7

document.write("<p>" + temp);

// Если аргумент отрицательный - он задает позицию относительно конца массива.

temp = a.slice(1, -1); // 2, 3, 4, 5, 6

document.write("<p>" + temp);

### Splice

Метод splice() - универсальный метод для вставки и удаления элементов из массива.

Метод возвращает массив из удаленных элементов.

arrayObj.splice( start, deleteCount, [elem1[, elem2[, ...[, elemN]]]] )

start - Индекс в массиве, с которого начинать удаление.

deleteCount - Кол-во элементов, которое требуется удалить, начиная с индекса start.

elem1, elem2, ..., elemN - Добавляемые элементы в массив. Добавление начинается с позиции start.

-->

<html>

<head>

<title>Метод splice()</title>

<script type="text/javascript">

var a = [1, 2, 3, 4, 5, 6, 7];

document.write("<p>Исходный массив: " + a);

// УДАЛЕНИЕ.

// splice (начальный индекс, сколько элементов удалить);

var d = a.splice(1, 3); // Будут удалены: 2, 3, 4

document.write("<p>Удалены: " + d);

document.write("<p>Остались: " + a);

// ВСТАВКА.

// splice (начальный индекс = 1, сколько элементов удалить = 0, значения для вставки с начального индекса = 2, 3, 4);

a.splice(1, 0, 2, 3, 4);

document.write("<p>После вставки: " + a);

### Sort

sort() - Сортирует элементы массива в алфавитном порядке

(при необходимости временно преобразуя элементы в строки для выполнения сравнения).

-->

<html>

<head>

<title>Метод sort() сортировка массива</title>

<script type="text/javascript">

var a = [1, 4, 11, 2, 9, 99, 5];

a.sort();

document.write(a.join("<p>"));

document.write("<hr/>");

var s = ["January", "February", "March", "April", "May"];

s.sort();

document.write(s.join("<p>"));

### Push

push() - добавление одного или нескольких элементов в конец массива.

(возвращаемое значение - количество элементов в новом массиве)

-->

<html>

<head>

<title>Метод Push()</title>

<script type="text/javascript">

var stack = ["January", "February", "March"];

document.write("<p>Исходный: " + stack.join(" "));

document.write("<p>Размер массива: " + stack.length);

document.write("<hr/>");

var lenght = stack.push("April");

document.write("<p>После добавления: " + stack.join(" "));

document.write("<p>Размер массива: " + lenght);

document.write("<hr/>");

lenght = stack.push("May", "June");

document.write("<p>После добавления: " + stack.join(" "));

document.write("<p>Размер массива: " + lenght);

### Pop

pop() - выполняет удаление последнего элемента массива.

(возвращаемое значение - удаленный элемент массива)

-->

<html>

<head>

<title>Метод Pop()</title>

<script type="text/javascript">

var stack = ["January", "February", "March", "April", "May", "June"];

document.write("<p>Исходный: " + stack.join(" "));

document.write("<p>Размер массива: " + stack.length);

document.write("<hr/>");

var lenght = stack.pop();

document.write("<p>После pop: " + stack.join(" "));

document.write("<p>Удаленный элемент: " + lenght);

document.write("<p>Размер массива после удаления: " + stack.length);

### Shift

shift() - удаляет и возвращает первый элемент массива.

-->

<html>

<head>

<title>Метод Shift()</title>

<script type="text/javascript">

var arr = [1, 2, 3];

document.write("<p>Исходный массив: " + arr.join(" "));

document.write("<hr/>");

var value = arr.shift();

document.write("<p>Массив после сдвига: " + arr.join(" "));

document.write("<p>Выдвинутое значение: " + value);

### Unshift

unshift() - добавляет элемент в начало массива.

-->

<html>

<head>

<title>Метод UnShift()</title>

<script type="text/javascript">

var arr = [1, 2, 3];

document.write("<p>Исходный массив: " + arr.join(" "));

document.write("<hr/>");

arr.unshift(0);

document.write("<p>Массив после сдвига: " + arr.join(" "));

document.write("<hr/>");

arr.unshift(-3, -2, -1);

document.write("<p>Массив после сдвига: " + arr.join(" "));

# Functions

## Functions

### Function

Функция - это блок кода, который можно многократно вызывать на исполнение.

При этом функции можно передавать аргументы (параметры) с которыми она будет работать.

Также можно вызывать функцию без параметров.

Для создания функции используется следующая семантика:

function имя\_функции ( аргументы ) { тело функции }

Вызов функции происходит следующим образом:

имя\_функции( аргументы );

-->

<html>

<head>

<title>Функции в JavaScript</title>

<script type="text/javascript">

function MyFirstFunction() {

document.write("<p>Hello from MyFirstFunction!");

document.write("<p>How are you?");

document.write("<p>Goodbye!");

}

function MySecondFunction() {

document.write("<p>Hello from MySecondFunction!");

}

// Вызов функций.

MyFirstFunction();

document.write("<hr/>");

MySecondFunction();

### Function args

var a = 10;

var b = 15;

var c = 20;

var d = 21;

function Sum1() {

var result = a + b;

document.write("<p>Sum1 result: " + result);

}

function Sum2(x1, x2) {

var result = x1 + x2;

document.write("<p>Sum2 result: " + result);

}

// Вызов функций.

Sum1();

Sum2(); // Не передаем параметров при вызове.

Sum2(c, d);

### Optional arg

function print(msg, count) { // Необязательный параметр count.

if (count == undefined) {

count = 3;

}

for (var i = 0; i < count; i++) {

document.write("<p>" + msg);

}

document.write("<hr/>");

}

// Вызов функций.

print("hello");

print("world", 7);

### Return

function Sum(a, b) {

return a + b;

}

var res = Sum(2, 3);

document.write(res);

### Array of arguments

При вызове функции создается специальный масив содержащий

в себе аргументы, которые были переданы при вызове.

У этого массива есть свойство length, в котором хранится

количество аргументов переданых функции.

-->

<html>

<head>

<title>Массив Аргументов arguments</title>

<script type="text/javascript">

function max(a, b, c) {

var maxValue = Number.NEGATIVE\_INFINITY;

document.write("<p>Number.NEGATIVE\_INFINITY = " + maxValue + "<hr/>");

for (var i = 0; i < arguments.length; i++)

if (arguments[i] > maxValue)

maxValue = arguments[i];

return maxValue;

}

var res = max(-33, 33, 777, -666, -22);

document.write("<p>Максимальное значение одного из аргументов функции = " + res);

## Scope

### Global and local variable

Область видимости - область, в которой переменная доступна для использования.

Локальная область видимости - тело функции, в которой объявлена переменная.

Глобальная область видимости - область видимости доступная для всех функций и сценариев.

-->

<html>

<head>

<title>Локальные и глобальные переменные</title>

<script type="text/javascript">

var global = "Глобальная переменная";

function f() {

var local = "Локальная переменная";

document.write("<p style='color:navy;'>" + global);

document.write("<p style='color:navy;'>" + local);

document.write("<hr/>");

}

f();

document.write("<p style='color:green;'>" + global);

// ОШИБКА: Попытка обращения к локальной переменной из-за предела тела функции f().

document.write("<p style='color:green;'>" + local);

### Global and local variable 2

var global = "Глобальная переменная";

// ВНИМАНИЕ: Создание глобальной переменной с именем как у локальной переменной из метода - f().

var local = "Глобальная переменная одноименная с локальной из метода - f().";

function f() {

// ВНИМАНИЕ: Создание локальной переменной с именем как у глобальной переменной.

var local = "Локальная переменная одноименная с глобальной переменной.";

document.write("<p style='color:navy;'>" + global);

document.write("<p style='color:gray;'>" + local);

document.write("<hr/>");

}

f();

document.write("<p style='color:green;'>" + global);

document.write("<p style='color:orange;'>" + local);

### No var

Если не создавать переменную явно через указание ключевого слова var и имени,

то при обращении по имени к еще не существующей переменной,

эта переменная будет создана автоматически как глобальная,

даже если обращение будет происходить локально внутри функции.

-->

<html>

<head>

<title>Отсутствие var</title>

<script type="text/javascript">

//var global = "Глобальная переменная";

function f() {

// Неявное создание глобальной переменной, через первое обращение на запись.

global = "Глобальная переменная";

document.write("<p style='color:navy;'>" + global);

document.write("<hr/>");

}

// ОШИБКА: Обращение на чтение до создания переменной в теле функции - f().

//document.write("<p style='color:green;'>" + global);

f();

// OK! Обращение на чтение после создания переменной в теле функции - f().

document.write("<p style='color:green;'>" + global);

### Local in functions

Тело функции - локальная область видимости.

ПРАВИЛО:

Из одной локальной области видимости нельзя обращаться к переменным

находящимся в другой локальной области видимости.

-->

<html>

<head>

<title>Область видимости функции</title>

<script type="text/javascript">

function A() {

var a = "Локальная переменная - a";

document.write("<p>" + a);

document.write("<p>" + a);

}

function B() {

var b = "Локальная переменная - b";

document.write("<p>" + b);

// ОШИБКА!

// Попытка обращения к локальной переменной находящейся в локальной области видимости метода A().

// document.write("<p>" + a);

}

A();

B();

### Cycle scope

Область видимости не ограничивается телом

условной или циклической конструкции.

-->

<html>

<head>

<title>Область видимости внутри конструкций</title>

<script type="text/javascript">

for (var counter = 0; counter < 3; counter++) {

document.write("<p style='background-color:lightblue;'>" + counter);

}

// counter - счетчик итераций цикла доступна вне тела цикла - for.

document.write("<p style='background-color:yellow;'>" + counter);

</script>

## Different function

### Recursion

<script type="text/javascript">

function f(counter) {

counter--;

document.write("<p style='background-color:lightblue;'>Первая половина метода: " + counter);

if (counter != 0)

f(counter);

document.write("<p style='background-color:yellow;'>Вторая половина метода: " + counter);

}

f(3);

### Literal function

<title>Функции-литералы</title>

<script type="text/javascript">

var f = function () {

document.write("<p>Hello!");

};

f(); // Вызов функции.

f = function () {

document.write("<p>Byе!");

};

f(); // Вызов функции.

### Function in function

<script type="text/javascript">

function outer() {

inner(); // Вызов функции.

function inner() { // Вложенная функция.

document.write("<p style='background-color:lightblue;'>Inner Function");

}

inner(); // Вызов функции.

}

outer(); // Вызов функции.

### Callback function

Функции передаваемые в качестве аргументов другой функции

(CallBack Function - функции обратного вызова).

-->

<html>

<head>

<title>CallBack Function</title>

<script type="text/javascript">

var add = function (a, b) {

return a + b;

};

var sub = function (a, b) {

return a - b;

};

function show(callBackFunction, a, b) {

var result = callBackFunction(a, b);

document.write("<p>" + result);

};

show(add, 10, 20);

show(sub, 50, 30);

# Objects

## Object creation

Объект – это составной тип данных,

он объединяет множество значений в единый модуль и

позволяет сохранять и извлекать значения по их именам.

Создание объекта через блок инициализации:

var имя\_обьекта = { имя\_свойства: значение\_свойства, имя\_свойства: значение\_свойства }

-->

<html>

<head>

<title>Обьекты</title>

<script type="text/javascript">

// Создание объекта через блок инициализации.

var instance = { propertyA: 222, propertyB: 444 };

// Вывод значений свойств.

document.write("<p>instance.propertyA = " + instance.propertyA);

document.write("<p>instance.propertyB = " + instance.propertyB);

document.write("<hr/>");

// Присвоение новых значений свойствам.

instance.propertyA = 333; // 1 способ.

instance["propertyB"] = 555; // 2 способ.

// Вывод значений свойств.

document.write("<p>instance.propertyA = " + instance.propertyA);

document.write("<p>instance.propertyB = " + instance.propertyB);

document.write("<hr/>");

document.write("<p>instance = " + instance);

## Object construction

Помимо создания объекта через блок инициализации, можно реализовать

создание через специально предусмотренный конструктор Object().

Конструктор не принимает никаких параметров,

а перед ним обязательно должно указываться ключевое слово new.

Свойства объекта будут создаваться автоматически при первом обращении

по имени будущего свойства и присвоения ему значения.

-->

<html>

<head>

<title>Конструктор Object()</title>

<script type="text/javascript">

// Создаем обьект с именем point.

var point = new Object();

// Обращение к свойству x обьекта point.

// Если свойство отсутствует то оно создается автоматически и ему присваивается значение.

point.x = 10;

// Создаем свойство y и присваиваем ему значение.

point.y = 15;

// Добавляем к объекту метод.

point.f = function () {

document.write("<p>x = " + point.x + "<p>y = " + point.y);

};

point.f();

## Object read write

К свойствам обьекта необходимо обращаться явно указывая имя обьекта,

символ "." и имя свойства.

Свойства можно создавать, читать, изменять и удалять из объекта.

-->

<html>

<head>

<title>Массивы в JavaScript</title>

<script type="text/javascript">

var person = {

firstName: "Jim",

lastName: "Rainbow",

age: 30,

occupation: "Web Developer",

show: function () {

document.write(this.firstName);

}

};

// Обращения к свойствам объекта.

document.write("<p>" + person.firstName + " " + person.lastName);

// Изменение значений свойств объекта.

person.age = 25;

document.write("<p>Age: " + person.age + " years old. </br>");

person.show();

## Вложенный объект

<html>

<head>

<title>Вложенные обьекты</title>

<script type="text/javascript">

var human = {

firstName: "Alex",

secondName: "Jazun",

age: 27,

// Вложенный обьект.

occupation: {

company: "Microsoft",

position: "Web Developer"

}

};

document.write("<p>First name: " + human.firstName);

document.write("<p>Second name: " + human.secondName);

document.write("<p>Age: " + human.age + " years old");

// Обращение к вложенному объекту.

document.write("<p>Company: " + human.occupation.company);

document.write("<p>Position: " + human.occupation.position);

## Outside property object

Вложеный обьект можно создать с помощью функции конструктора.

-->

<html>

<head>

<title>Создание вложеных обьектов через конструктор</title>

<script type="text/javascript">

var human = new Object();

human.firstName = "Alex";

human.secondName = "Jazun";

human.age = 27;

// Создание вложенного обьекта.

human.occupation = new Object();

human.occupation.company = "Microsoft";

human.occupation.position = "Web Developer";

document.write("<p>First name: " + human.firstName);

document.write("<p>Second name: " + human.secondName);

document.write("<p>Age: " + human.age + " years old");

// Обращение к вложенному объекту.

document.write("<p>Company: " + human.occupation.company);

document.write("<p>Position: " + human.occupation.position);

## Methods

Метод - это функция принадлежащая конкретному объекту,

которую вызвать ее можно только на этом объекте.

При этом такая функция описывается, как функция литерал присвоенная свойству.

method: function() {.....}

-->

<html>

<head>

<title>Методы</title>

<script type="text/javascript">

var human = {

firstName: "Alex",

secondName: "Jazun",

age: 27,

occupation: "<img src='Images/Microsoft.jpg' width='152' height='38'/>",

// Метод show() в блоке инициализации обьекта.

show: function () {

document.write("<p>First name: " + this.firstName);

document.write("<p>Second name: " + this.secondName);

document.write("<p>Age: " + this.age + " years old<br/>");

document.write(this.occupation + "<hr/>");

},

// Создаем метод SayGoodbye() в блоке инициализации обьекта

changeWork: function (occupation) {

this.occupation = occupation;

}

}

// Вызов методов.

human.show();

human.changeWork("<img src='Images/CyberBionic Systematics.jpg'/>");

human.show();

</script>

## Outside methods

var builder = new Object();

builder.createRoof = function () {

document.write("<img src='Images/Roof.gif'/><br/>");

};

builder.createFirstFloor = function () {

document.write("<img src='Images/FirstFloor.gif'/><br/>");

};

builder.createSecondFloor = function () {

document.write("<img src='Images/SecondFloor.gif'/><br/>");

};

builder.createBasement = function () {

document.write("<img src='Images/Basement.gif'/><br/>");

};

// Строительство дома.

builder.createRoof();

builder.createSecondFloor();

builder.createFirstFloor();

builder.createBasement();

For in cycle

Циклическая конструкция for in, используется для перебора

элементов массива или перебора свойств объекта.

Принцип работы конструкции:

for ( переменная\_итерации in массив\_или\_обьект) { ...тело конструкции ...}

-->

<html>

<head>

<title>Конструкция for in</title>

<script type="text/javascript">

var house = {

roof: "<img src='Images/Roof.gif'/><br/>",

secondFloor: "<img src='Images/SecondFloor.gif'/><br/>",

firstFloor: "<img src='Images/FirstFloor.gif'/><br/>",

basement: "<img src='Images/Basement.gif'/><br/>",

address: "<p style='color:green'>UK. London. Muswell Hill. N10<hr/>"

}

// Перебор значений свойств в обьекте - house

for (element in house) {

document.write(house[element]);

}

## Property exist

Конструкция if(имя\_свойства in объект) -

используется для проверки наличия свойства у объекта.

-->

<html>

<head>

<title>Проверка существования свойства</title>

<script type="text/javascript">

var house = {

roof: "<img src='Images/Roof.gif'/><br/>",

// thirdFloor: отсутствует.

secondFloor: "<img src='Images/SecondFloor.gif'/><br/>",

firstFloor: "<img src='Images/FirstFloor.gif'/><br/>",

basement: "<img src='Images/Basement.gif'/><br/>",

address: "<p style='color:green'>UK. London. Muswell Hill. N10<hr/>"

}

// Строительство дома.

document.write(house.roof);

// ЕСЛИ: В объекте house имеется планировка третьего этажа - thirdFloor,

if ("thirdFloor" in house) { // ТО: Строим третий этаж с планировкой - thirdFloor.

document.write(house["thirdFloor"]);

}

else { // Иначе: Строим третий этаж с планировкой - secondFloor.

document.write(house["secondFloor"]);

}

document.write(house.secondFloor);

document.write(house.firstFloor);

document.write(house.basement);

document.write(house.address);

## Delete property

delete - оператор который способен удалить свойство в обьекте,

для этого необходимо использовать его следующим образом:

delete имя\_обьекта.имя\_свойства;

При обращении к свойству впоследствии будет возвращаться undefined.

-->

<html>

<head>

<title>Оператор delete</title>

<script type="text/javascript">

var house = {

roof: "<img src='Images/Roof.gif'/><br/>",

secondFloor: "<img src='Images/SecondFloor.gif'/><br/>",

firstFloor: "<img src='Images/FirstFloor.gif'/><br/>",

basement: "<img src='Images/Basement.gif'/><br/>",

address: "<p style='color:green'>UK. London. Muswell Hill. N10<hr/>"

}

// Строительство дома.

if ("secondFloor" in house) {

// Удаление свойства secondFloor из объекта house.

delete house.secondFloor;

}

for (element in house) {

document.write(house[element]);

}

## Delete element

С помощью оператора delete можно удалять

элементы массивов по указанному индексу.

-->

<html>

<head>

<title>Оператор delete в масивах</title>

<script type="text/javascript">

var array = new Array(0, 1, 2, 3, 4, 5, 6, 7);

document.write("<p>Массив до удаления: " + array.join(", "));

var index = prompt("Выберите индекс элемента для удаления: " + array.join(", "));

// Удаление элемента масива по указанному индексу.

delete array[index];

document.write("<p>Массив после удаления: " + array.join(", "));

## With

Оператор with создает новую область видимости и

представляет свойства объекта «object» как локальные переменные.

Оператор with является устаревшим и его использование не желательно.

Его использование запрещено в Strict Mode. Подробнее: http://habrahabr.ru/post/118666/

-->

<html>

<head>

<title>Обьекты, создание</title>

<script type="text/javascript">

var house = {

roof: "<img src='Images/Roof.gif'/><br/>",

secondFloor: "<img src='Images/SecondFloor.gif'/><br/>",

firstFloor: "<img src='Images/FirstFloor.gif'/><br/>",

basement: "<img src='Images/Basement.gif'/><br/>",

address: "<p style='color:green'>UK. London. Muswell Hill. N10<hr/>"

}

with (house) {

document.write(roof);

document.write(secondFloor);

document.write(firstFloor);

document.write(basement);

}