# Literals and Constructors

## Object constructor

// Конструктор Object может принимать параметр и делегировать вызов другому

// встроенному конструктору, вернув в результате объект другого типа.

var obj = new Object();

console.log(obj.constructor === Object); // true

var obj = new Object(1);

console.log(obj.constructor === Number); // true

console.log(obj.toFixed(3)); // 1.000

var obj = new Object("Hello world");

console.log(obj.constructor === String); // true

var obj = new Object(true);

console.log(obj.constructor === Boolean); // true

var a = 100; // простое число

var b = new Number(100); // объект со значением 100

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

document.write(typeof b + "<br />"); // object

var str = "hello world"; // простое значение

// при вызове toUpperCase() строка временно преобразовывается в объект String

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

// свойства и методы можно вызывать непосредственно на значении

document.write("hello".length);

# Functions

## Scope

// В JavaScript есть две области видимости:

// глобальная - представленная объектом window (если среда выполнения сценария - браузер)

// локальная - представленна функцией.

// операторные скобки { и } не создают обалсть видимости как в других языках программирования.

function myfunction() {

// var i, test;

// переменные i и test доступны в начале функции, хотя определяються далее в теле функции.

console.log(i); // undefined

console.log(test); // undefined

for (var i = 0; i < 10; i++) {

var test = i;

}

console.log(i); // 10

console.log(test); // 9

}

myfunction();

## Callback call

// Для того что бы функция была максимально универсальной она только возвращает узлы документа не изменяя их.

function findNodes(callback, callbackObj) {

var nodes = [],

pArray = [],

found;

pArray = document.getElementsByTagName("p");

if (typeof callback != "function") {

callback = false;

}

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

found = pArray[i];

if (callback) {

// вызов функции на объекте callbackObj

callback.call(callbackObj, found);

}

nodes.push(found);

}

return nodes;

}

var app = {};

app.color = "green";

app.borderStyle = "solid";

app.borderWidth = "1px";

// Функция для изменения найденного узла.

app.makeBorder = function (node) {

node.style.border = this.borderStyle + " " + this.borderWidth + " " + this.color;

}

// найти узлы и добавить им рамку

findNodes(app.makeBorder, app);

## **Immediately call function**

var obj = {

// немедленно вызываемая функция может использоваться при задании значений

// свойств объектов.

name: (function () {

var fName = "Ivan",

lName = "Ivanov";

return fName + " " + lName

}()),

sayHello: function () {

alert("Hello my name is " + this.name);

}

};

## **AddListener**

var utils = {

addListener: null,

removeListener: null

};

if (typeof window.addEventListener === "function") {

utils.addListener = function (e, type, fn) {

e.addEventListener(type, fn, false);

}

utils.removeListener = function (e, type, fn) {

e.removeEventListener(type, fn, false);

}

}

else if (typeof document.attachEvent === "function") {

utils.addListener = function (e, type, fn) {

e.attachEvent("on" + type, fn);

}

utils.removeListener = function (e, type, fn) {

e.detachEvent("on" + type, fn);

}

}

else {

utils.addListener = function (e, type, fn) {

e["on" + type] = fn;

}

utils.removeListener = function (e, type, fn) {

e["on" + type] = null;

}

}

// Реализация методов объекта utils не оптимальна, так как при каждом вызове addListener или removeListener

// выполняется один и тот же набор проверок.

utils.addListener(window, "load", function () {

var testBtn = document.getElementById("test"),

removeBtn = document.getElementById("removeHandler");

function say() {

alert("Hello world");

}

utils.addListener(testBtn, "click", say)

utils.addListener(removeBtn, "click", function () {

utils.removeListener(testBtn, "click", say);

})

});

## **Closure**

<style>

body {

font-family: Helvetica, Arial, sans-serif;

font-size: 12px;

}

h1 {

font-size: 1.5em;

}

h2 {

font-size: 1.2em;

}

</style>

</head>

<body>

<a href="#" id="size-16">16</a>

<a href="#" id="size-20">20</a>

<a href="#" id="size-24">24</a>

<script>

function makeSizer(size) {

return function () {

document.body.style.fontSize = size + 'px';

};

}

// замыкания с различными значениями переменной size

var size16 = makeSizer(16);

var size20 = makeSizer(20);

var size24 = makeSizer(24);

document.getElementById('size-16').onclick = size16;

document.getElementById('size-20').onclick = size20;

document.getElementById('size-24').onclick = size24;

## **Memoize**

// Мемоизация - прием кэширования результатов функции.

function calcFib(x) {

// проверка наличия результата для парамтра x.

if (!calcFib.cache[x]) { // если результат не найден то производим вычесления.

if (x > 1) {

calcFib.cache[x] = calcFib(x - 1) + calcFib(x - 2);

}

else {

calcFib.cache[x] = x;

}

}

return calcFib.cache[x];

}

// создание свойства в функции с пустым объектом.

calcFib.cache = {};

for (var i = 0; i < 40; i++) {

document.write(i + " = " + calcFib(i) + "<br />");

}

## Apply

window.onload = function () {

var person = {

firstName: "Ivan",

lastName: "Ivanov"

};

function print(color, count) {

var p = document.getElementById("output");

p.style.color = color;

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

p.innerHTML += "Hello, my name is " + this.firstName + "<br />";

}

}

// Применение функции для объекта person

// Метод apply вызывается на функции и в качестве первого параметра принимает объект, на который будет ссылаться ключевое слово this

// в теле функции, а в качестве второго параметра массив аргументов, который будет передан в функцию.

// В данном случае функция print вызывается как метод объекта person.

print.apply(person, ["green", 3]);

// Метод call аналогичен методу apply за исключением того, что аргументы передаются не массивом, а отдельными параметрами.

//print.call(person, "green", 3);

}

</script>

</head>

<body>

<p id="output">

</p>

</body>

## Max in array

var arr = [1, 4, 7, 2, 34, 734, 1, 33, 667, 3, 56, 778, 1, 80];

var maxValue = Math.max.apply(null, arr); // поиск максимального значения в массиве.

// вызов метода Math.max.apply(null, arr) идентичен вызову Math.max(arr[0], arr[1], arr[2], arr[3], arr[4], arr[5]);

alert(maxValue);

## Curry

function curry(fn) {

var slice = Array.prototype.slice,

oldArgs = slice.call(arguments, 1); // отбрасываем первый аргумент

return function () {

var newArgs = slice.call(arguments),

args = oldArgs.concat(newArgs);

return fn.apply(null, args); // вызываем функцию для, которой проводилось каррирование

}

}

function add(x, y) {

return x + y;

}

var add10 = curry(add, 10);

var r1 = add10(5); // 15

var r2 = add10(10); // 20

document.write(r1 + "<br />");

document.write(r2 + "<br />");

function sum(a, b, c) {

return a + b + c;

}

var newSum = curry(sum, 1, 2);

var r3 = newSum(5); // 8

document.write(r3);

# Namespaces

// Перед тем как создавать новый глобальный объект и использовать его как пространство имен,

// следует убедиться, что такого объекта еще нет.

// Если объект уже существует то повторная его инициализация уничтожит весе свойства, которые

// присутствовали в объекте до создания нового пространства имен.

var App = {} // не безопасный код

// безопасный код

if (typeof App === "undefined") {

var App = {};

}

// или

var App = App || {};

# DOM Patterns

## Browser type

/ АНТИШАБЛОН - использование имени браузера для того что бы проверить его возможности.

if (navigator.userAgent.indexOf("MSIE") !== -1) {

document.attachEvent("onclick", function () {

console.log("clicked");

});

}

// Предпочтительно

if (document.attachEvent) {

document.attachEvent("onclick", function () {

console.log("clicked");

});

}

if (typeof document.attachEvent !== "undefined") {

document.attachEvent("onclick", function () {

console.log("clicked");

});

}

Loop inner html

// АНТИШАБЛОН

for (var i = 0; i < 100; i++) {

// обращаться к DOM на каждой итерации цикла не оптимально.

document.getElementById("test").innerHTML += i + ", ";

}

// Предпочтительней

var i, msg = "";

for (i = 0; i < 100; i++) {

msg += i + ", ";

}

document.getElementById("test").innerHTML += msg;

## **QuerySelector**

var div1 = document.getElementById("test");

div1.innerHTML = "Hello world";

// в большинстве случаев функции querySelector работают быстрее чем другие функции для получения элементов из DOM

var div2 = document.querySelector("#test");

div2.innerHTML = "Hello world";

## DocumentFragment

var p, t, fragment;

// Doument Fragment - фрагмент документа в который можно добавлять дочерние элементы. При этом дочерние элементы

// фрагмента не будут отображаться на странице до тех пор пока фрагмент не станет дочерним элементом существующего элемента DOM.

fragment = document.createDocumentFragment();

p = document.createElement("p");

t = document.createTextNode("First paragraph");

p.appendChild(t);

fragment.appendChild(p); // добавление элемента в фрагмент

p = document.createElement("p");

t = document.createTextNode("Second paragraph");

p.appendChild(t);

fragment.appendChild(p); // добавление элемента в фрагмент

document.body.appendChild(fragment); // добавление фрагмента в тело документа.

// все элементы которые есть в фрагменте станут дочерними элементами документа.

## Worker

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Использование Web Workers</title>

<script src="fibonachiWorker.js"></script>

<script>

window.onload = function () {

document.getElementById("startButton").onclick = function () {

// Создание нового потока

var worker = new Worker("fibonachiWorker.js");

// Создание обработчика для получения сообщений от потока

worker.addEventListener("message", function (e) {

document.getElementById("output").innerHTML += e.data + "<br />";

}, false);

// запуск потока

worker.postMessage("");

}

}

</script>

</head>

<body>

<button id="startButton">Start</button>

<p id="output"></p>

</body>

</html>

function calcFib(x) {

if (x > 1) {

return calcFib(x - 1) + calcFib(x - 2);

}

else {

return x;

}

}

addEventListener("message", function () {

for (var i = 0; i < 50; i++) {

postMessage(i + " = " + calcFib(i) + "<br />");

}

}, false);

## Script files

(function () {

var testVar1, testVar2, testVar3;

for (var i = 0; i < 100; i++) {

testVar1 = i;

testVar2 = i + 1;

testVar3 = i \* 3;

}

})();

(function () {

alert('test');

})();

(function () {

function myfunction() {

}

function veryLongName() {

}

function veryVeryVeryLongName() {

}

myfunction();

veryLongName();

veryVeryVeryLongName();

})();

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Развёртывание сценариев</title>

</head>

<body>

<h1>Данное сообщение будет отображено <span style="color:red">до</span> загрузки и выполнения всех сценариев</h1>

<!--

Для зжатия JS файлов можно использовать следующие инструементы:

http://jscompress.com/

http://closure-compiler.appspot.com/home

http://dean.edwards.name/packer/

http://refresh-sf.com/yui/

-->

<script src="JavaScript\_date.js"></script>

</body>

</html>