MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

O. M. BEKETOV NATIONAL UNIVERSITY
OF URBAN ECONOMY in KHARKIV

B. P. Bocharov, Y. V. Levikov, M. Y. Voevodina

SCRIPTING PROGRAMMING LANGUAGES

TUTORIAL

Kharkiv – O. M. Beketov NUUE – 2021

УДК 004.92(075.8)
Б86

Authors:

Bocharov Boris Petrovich, *candidat of sciences techniques, dotsent;*
Levikov Yurii Volodymyrovych, *lecturer;*
Voevodina Maria Yuriivna, *lecturer*

Reviewers:

O. V. Gritsunov, Dr. of sciences techniques, professor, prof. Department of microelectronics, electronic devices and devices of Kharkiv National University of Radio Electronics;

O. V. Pankratov, Dr. of sciences techniques, senior researcher of Institute of Mechanical Engineering

Recommended for publication
Academic Council of O. M. Beketov NUUE in Kharkiv as a tutorial
(protocol № 5 dated November 30, 2018)

JavaScript є мовою програмування для Веб. Переважна більшість веб-сайтів використовують JavaScript, а всі сучасні веб-переглядачі – для настільних комп'ютерів, ігрових консолей, електронних планшетів та смартфонів – містять інтерпретатор JavaScript, що робить JavaScript найбільш широко використовуваною мовою програмування, яка колись існувала в історії.

Bocharov B. P.

Б86 Scripting programming languages : tutorial / B. P. Bocharov, Y. V. Levikov, M. Y. Voevodina ; O. M. Beketov National University of Urban Economy in Kharkiv. – Kharkiv : O. M. Beketov NUUE in Kharkiv, 2021. – 109 p.

JavaScript is programming language for Web. The vast majority of websites use JavaScript, and all modern web browsers – for desktops, game consoles, electronic tablets and smartphones – include a JavaScript interpreter, which makes JavaScript the most widely used programming language that have ever existed in history.

УДК 004.92(075.8)

Content

Introduction

JavaScript is programming language for Web. The vast majority of websites use JavaScript, and all modern web browsers – for desktops, game consoles, electronic tablets and smartphones – include a JavaScript interpreter, which makes JavaScript the most widely used programming language that have ever existed in history. JavaScript is one of three technologies that any web developer should know:

– HTML markup language that allows you to define the content of web pages,

– CSS style language, which allows you to define the appearance of web pages,

– The programming language JavaScript, which allows you to determine the behavior of web pages.

If you know other programming languages, you may find information useful that JavaScript is a high-level, dynamic, untyped and interpreted programming language that is well suited for programming in object-oriented and functional styles.

Its JavaScript syntax inherited from the Java language, its first-class functions – from the Scheme language, and the mechanism of inheritance on the basis of prototypes – from the language of Self.

The name of the language "JavaScript" can be misleading. With the exception of surface syntactic similarity, JavaScript is completely different from the Java programming language.

JavaScript has long outgrown the framework of the scripting language, turning into a reliable and effective universal programming language.

JavaScript was created in the company Netscape at the dawn of the Web. The name "JavaScript" is a trademark registered by Sun Microsystems (now Oracle), and is used to describe the implementation of the language created by Netscape (now Mozilla). Netscape has introduced a language for standardization of the European Computer Manufacturer's Association (ECMA), but because of legal trademark problems, the standardized version of the language has received a somewhat awkward name, ECMAScript.

Due to the same legal problems, the language version from Microsoft received the official name "JScript". However, in practice, all these implementations are usually called JavaScript

During the last decade, all web browsers provided implementation of version 3 of the ECMAScript standard, and in fact, developers did not need to think about version numbers: the language standard was stable, and its implementation in web browsers was largely compatible.

Recently, a new important version of the standard language called ECMAScript 5 has released.

The fourth version of the ECMAScript standard was developed for many years, but because of too ambitious goals it was never released.

Google is developing its JavaScript interpreter called V8. It is based on the version of node js.

Part 1 Basic Concepts of the JavaScript

A brief overview of the JavaScript language.

```
// Everything that follows two slashes is a comment.
// Attentively read the comments: they describe the JavaScript code.
// Variables are declared using the keyword var:
var x; // Declaring a variable named x
// Assign values to variables using the =
x = 0; // Now the variable x has the value 0
// x => 0: In expressions, the name of the variable is replaced by its value.
// JavaScript supports values of different types
x = 1; // Numbers.
x = 0.01; // Integer and real numbers are represented by one type.
x = "hello world"; // Strings of text in quotes.
x = 'JavaScript'; // Strings can also be enclosed in apostrophes.
x = true; // Boolean values.
x = false; // Another Boolean value.
x = null; // null is a special value, indicating as "no value".
x = undefined; // The value of undefined is similar to null
```

Two other very important types of data that can be manipulated by JavaScript programs are objects and arrays. They will be considered later.

```
// The most important type of data in JavaScript are objects.
// An object is a collection of name / value pairs or a mapping of a string to a value.
var book = {            // The objects are enclosed in braces.
topic: "JavaScript", // The property "topic" is set to "JavaScript".
fat: true                 // The property "fat" is set to true.
};                          // The braces mark the end of the object.
// Access to object properties using '.' or '[]' :
book.topic          // => "JavaScript"
book["fat"]          // => true: another way to get the property value.
```

```
book.author = “Ivanov”; // Create a new property by assignment.
book.contents = {};        // {} - empty object
// JavaScript supports arrays (lists with numeric indices) of values:
var primes = [2, 3, 5, 7]; // An array of 4 values is limited to [ and ].
primes[0]                  // => 2: первый элемент (с индексом 0) массива.
primes.length              // => 4: number of elements in the array.
primes[primes.length-1]  // => 7: the last element of the array.
primes[4] = 9;             // Add a new element assignment.
primes[4] = 11;     // Or change the value of an existing item.
var empty = [];     // [] – empty array without ellements.
empty.length        // => 0
// Arrays and objects can store other arrays and objects:
var points = [        // Array with 2 elements.
        {x:0, y:0},   // Each element is an object.
        x:1, y:1}
];
var data = {           // Object with 2 properties.
 trial1: [[1,2], [3,4]], // The value of each property is an array.
 trial2: [[2,3], [4,5]] // The elements of the array are arrays.
};
```

Syntactic constructs that contain lists of array elements in square brackets or display object properties in values within braces are often called initialization expressions.

An expression is a phrase in the JavaScript language that can be calculated to get a value. For example, use ‘.’ and ‘[]’ to reference the value of the property of an object or array element is an expression.

The most common way of generating expressions in JavaScript is to use operators.

```
// Operators perform actions with values (operands) and reproduce the new value.
// The most commonly used are arithmetic operators:
3 + 2          // => 5: addition
3 - 2          // => 1: subtraction
```

```
3 * 2           // => 6: multiplication
3 / 2           // => 1.5: division
points[1].x - points[0].x  // => 1: you can use more complex operands
"3" + "2"               // => "32": + adds numbers, combines strings
// In JavaScript, there are some abbreviated forms of arithmetic operators
var count = 0;       // Declaration of variable
count++;             // Increase the value of the variable by 1
count--;             // Decreasing the value of the variable by 1
count += 2;          // Add 2: the same as count = count + 2;
count *= 3;          // Multiply by 3: same as count = count * 3;
count                // => 6: names of variables themselves are expressions
// Comparison operators allow you to check two values for equality
// or inequality, find out which value is less or more, etc.
// They return true or false.
var x = 2, y = 3;    // Signs '=' perform assignment, not comparison
x == y               // => false: equality
x!= y          // => true: inequality
x < y                // => true: less
x <= y               // => true: less or equally
x > y                // => false: more
x >= y               // => false: more or equally
"two" == "three"   // => false: two different strings
"two" > "three"     // => true: when sorting alphabetically the strings
// "tw" more than "th"
false == (x> y) // => true: false equally false
// Logical operators combine or invert logical values
(x == 2) && (y == 3)      // => true: both comparisons are true. && - "and"
(x > 3) || (y < 3)              // => false: both comparisons are false. || - "or"
! (x == y) // => true: ! inverts a Boolean value
```

A function is a named and parameterized block of JavaScript code that is defined once, and can be used multiple times.

Here are some examples of simple functions:

```
function plus1(x) { // Define a function named "plus1" and with the parameter "x"
    return x+1;  // Return value 1 more than received
} // Functions are enclosed in braces
plus1(y)      // => 4: y has a value of 3, so this call will return 3 + 1
var square = function(x) { // Functions can be assigned to variables
 return x*x;  // Calculate the value of the function
};      // A semicolon marks the end of the assignment.
square(plus1(y))    // => 16: call two functions in one expression
```

When you combine functions with objects, you get the following methods:

```
// The functions assigned to the properties of objects are called methods.
// All objects in JavaScript have methods:
var a = [];              // Create an empty array
a.push(1,2,3);        // The push () method adds elements to the array
a.reverse();           // Another method: rearranges the elements in the reverse order
// You can define your own methods.
// The keyword "this" refers to the object in which the method is defined: in this case,
the array points.
points.dist = function() { // Method for calculating the distance between points
    var p1 = this[0];    // The first element of the array, relative to which the method
    is called
    var p2 = this[1]; /          // The second element of the "this"
    var a = p2.x-p1.x;          // Difference of coordinates X
    var b = p2.y-p1.y;          // Difference of coordinates Y
    return Math.sqrt(a*a + b*b); // Math.sqrt() calculates the square root
};
points.dist() // => 1.414: distance between 2 points
```

Consider several functions that demonstrate the application of the most commonly used JavaScript control instructions:

```
// JavaScript has conditional instructions and instructions of cycles syntactically similar
to similar instructions C, C ++, Java and other languages.
function abs(x) {           // The function that calculates the absolute value
if (x >= 0) {               // instructions if …
return x;      // executes this code if the comparison yields true.
}       // the end of instructions if
else {              // An optional else clause executes its code,
return -x;          // if the comparison evaluates to false.
} // Braces can be omitted if the sentence contains 1 statement.
function factorial(n) {     // The function that calculates the factorial
   var product = 1;         // Start with a product equal to 1
   while(n > 1) {   // Repeat the instructions in {} while expressing in () true
      product *= n;         //The shortened form of the expression product = product*n
      n--;                  // The abbreviated form of the expression n = n – 1
   }                // the end of cycle
   return product;
}
factorial(4) // => 24: 1*4*3*2
function factorial2(n) { // Another version that uses a different cycle
var i, product = 1; // Start from 1
for(i=2; i <= n; i++)       // i automatically increases from 2 to n
      product *= i;               // Perform in each cycle {} can be omitted,
      return product; // return factorial
}
factorial2(5) // => 120: 1*2*3*4*5
```

JavaScript is an object-oriented language, but the object model used in it differs radically from the model used in most other languages.

Below is a very simple example demonstrating the definition of the JavaScript class for representing points on a plane. Objects that are instances of this class have a single method r () that computes the distance between a given point and the origin:

```
// Define a constructor function to initialize a new object Point
function Point(x,y) {       //By agreement, the name of the constructors
                              // begins with the capital letter
this.x = x;             // this - reference to the initialized object
this.y = y;             // Save arguments in object properties
} // Nothing is returned
// To create a new instance, you must call the constructor function with the keyword
"new"
var p = new Point (1, 1); // Point on the plane with coordinates (1,1)
// Methods of objects Point are determined by assigning functions
// to the properties of the prototype object associated with the constructor function.
Point.prototype.r = function () {
// Return the square root of x2 + y2
// this is a Point object, against which the method is invoked.
   return Math.sqrt( this.x * this.x +  this.y * this.y );
};
// Now object p of type Point (and all subsequent Point objects)
//inherits the method r()
p.r () // => 1.414 ...
```

Lexical Structure

The lexical structure of a programming language is a set of elementary rules that determine how programs are written in that language. This is a low-level syntax for the language; it defines the kind of variable names, the symbols used to denote comments, and how one instruction separates from the other.

Character set

When writing programs in JavaScript, the Unicode character set is used. Unicode is a superset of ASCII and Latin-1 encodings and supports almost all written languages on the planet. The ECMAScript 3 standard requires that JavaScript implementations support the Unicode standard version 2.1 or higher, and the ECMAScript 5 standard requires that implementations provide support for the Unicode version 3 or higher.

Case sensitivity

JavaScript is a language that is case sensitive. This means that keywords, variable and function names, and any other language identifiers must always contain the same set of uppercase and lowercase letters. For example, the keyword while must be typed as "while", not "While" or "WHILE". Similarly online, Online, OnLine and ONLINE are names of four different variables.

Note, however, that HTML markup language (unlike XHTML) is not case sensitive. As HTML and client JavaScript are tightly coupled, this difference can lead to confusion. Many JavaScript objects and their properties have the same names as HTML tags and attributes, which they denote. However, if in HTML these tags and attributes can be typed in any register, then in JavaScript they usually should be typed in lowercase letters. For example, the attribute onclick of the event handler is most often set in HTML as onClick, however, in JavaScript code (or in an XHTML document), it must be marked as onclick.

Spaces, line breaks and format control characters

JavaScript ignores spaces that can be present between tokens in the program. In addition, JavaScript also for the most part ignores line feed characters. Therefore, spaces and newlines can be used without restriction in the source code of the programs for formatting and giving them a human-readable appearance. In addition to the usual space character (\ u0020), JavaScript also recognizes as whitespace the following characters: tab (\ u0009), vertical tab (\ u000B), format translation (\ u000C), non-breaking space (\ u00A0), byte order marker (\ uFEFF), as well as all Unicode characters belonging to the category Zs. The following characters are recognized by the Javascript interpreter as end-of-line characters: line feed (\ u000A), carriage return (\ u000D), line separator (\ u2028) and paragraph separator (\ u2029). A sequence of carriage return and linefeed characters is interpreted as a single line termination character. Unicode characters that control the format (category Cf), such as

RIGHT-TO-LEFT MARK (\ u200F) and LEFT-TO-RIGHT MARK (\ u200E), control the visual representation of the text in which they are present. They are important for the correct display of text in some languages and are valid in JavaScript comments, string literals, and regular expression literals, but not in identifiers (such as variable names) defined in JavaScript programs. The exceptions are ZEROWIDTH JOINER (\ u200D) and ZERO WIDTH NON-JOINER (\ u200C), which you can use in identifiers, provided that they are not the first characters of identifiers. As noted above, the byte order control character (\ uFEFF) is interpreted as a whitespace character.

Shielded Unicode sequences can also appear in comments, but since comments are ignored, in this context they are treated as a sequence of ASCII characters and are not interpreted as Unicode characters.

Normalization

Unicode allows you to encode the same character in several ways. These presentation methods provide the same display in a text editor, but they have different binary codes and are considered different from the computer's point of view. The Unicode standard specifies the preferred encoding methods for all characters and specifies a normalization procedure for converting the text to a canonical form suitable for comparison. JavaScript interpreters believe that the interpreted code has already been normalized, and does not attempt to normalize identifiers, strings, or regular expressions.

Comments

JavaScript supports two ways of commenting. Any text between // and the end of a line is treated as a comment and ignored by JavaScript. Any text between the / * and * / characters is also treated as a comment. These comments can consist of several lines, but can not be nested. The following lines are valid JavaScript comments:

```
// This is a one-line comment.

/ * This is also a comment * / // and this is another comment.

/ *

* This is another comment.

* It is located in several lines.

* /
```

Literals

A literal is the value specified directly in the text of the program. The following are examples of different literals:

12 // Number twelve

1.2 // The number one is two-tenths

"hello world" // Text line

'Hi' // Another line

true // Boolean

false // Other Boolean value

/ javascript / gi // The literal of the "regular expression" (for searching by pattern)

null // Empty object

Below are more complex expressions that can serve as literals for arrays and objects:

{x: 1, y: 2} // Object initializer

[1,2,3,4,5] // Array initializer

Identifiers and reserved words

An identifier is just a name. In JavaScript, identifiers act as the names of variables and functions, as well as labels for some loops. Identifiers in JavaScript must begin with a letter, with an underscore (_) or a dollar sign ($).Any letters, numbers, underscores or dollar signs can follow. (A digit can not be the first character, since then it will be difficult for an interpreter to distinguish identifiers from numbers). Examples of valid identifiers:

i

my_variable_name

v13

_dummy

$str

For compatibility and ease of editing, only ASCII characters and numbers are usually used to compose identifiers. However, JavaScript allows the use of letters and digits from the full set of Unicode characters in identifiers. (Technically, the ECMAScript standard also allows Unicode characters from the Mn, Mc and Pc categories in the identifiers, provided they are not the first characters of the identifiers.) This allows programmers to give names to variables in their native languages and use mathematical symbols in them.

JavaScript reserves a number of identifiers that play the role of keywords of the language itself. These keywords can not serve as identifiers in programs:

break	finally	this
case	for	throw
catch	function	true
continue	if	try
debugger	in	typeof
default	instancejof	var
delete	new	void
do	null	while
else	return	with
false	switch	

Data Types, Values, and Variables

During operation, computer programs manipulate values, for example the number 3.14 or the text "Hello World". Data types The types of values that can be represented and processed in a programming language are known as data types, and one of the most fundamental characteristics of any programming language is the set of data types it supports.

When a program needs to save a value to use it later, this value is assigned (or stored in) a variable. A variable defines a symbolic name for a value and provides the ability to get this value by name. The principle of the action of variables is another fundamental characteristic of any language.

The types of data in JavaScript can be divided into two categories: simple types and objects. The category of simple types in the JavaScript language includes numbers, text strings (which are usually called just strings) and logical (or Boolean) values.

The special values “null” and “undefined” are elementary values, but they do not apply to numbers, or to strings, or to boolean values. Each of them defines only one value of its own special type.

Any value in JavaScript that is not a number, a string, a Boolean value, or a special value, “null” or “undefined”, is an object.

An object (that is, a member of an object data type) is a collection of properties, each of which has a name and value (either a simple type, such as a number or string, or an object type).

Numbers

Unlike many programming languages, JavaScript does not distinguish between integers and real values. All numbers in JavaScript are represented by real values (floating point).

To represent numbers in JavaScript, a 64-bit format is used, defined by the IEEE 754.1 standard.

This format is known to programmers in the Java language as a double type. The same format is used to represent numbers of type double in most modern implementations of C and C ++.

This format is able to represent numbers in the range of ±1.7976931348623157x10308 to ± 5x10-324.

The format of the representation of real numbers in JavaScript allows you to accurately represent all integers from -9007199254740992 (-253) to 9007199254740992 (253) inclusive. For integer values outside this range, precision in the lower order bits may be lost.

It should be noted that some operations in JavaScript (such as accessing array elements by indices and bit operations) are performed with 32-bit integer values.

The number that is directly in the program in the JavaScript language is called a numeric literal.

Arithmetic operations in JavaScript

The processing of numbers in the JavaScript language is performed using arithmetic operators. The number of such operators includes: the addition operator +, the subtraction operator -, the multiplication operator *, the division operator / and the modulo operator% (returns the remainder of the division). A full description of these and other operators can be found in the reference.

In addition to these simple arithmetic operators, JavaScript supports more complex mathematical operations, using functions and constants available as properties of the Math object.

Arithmetic operations in JavaScript do not raise an error in case of overflow, loss of significant digits or division by zero. If the result of the arithmetic operation is greater than the largest representable value (overflow), the special value "infinity", which in JavaScript is denoted as Infinity, is returned. Similarly, if the absolute value of the negative result is greater than the largest representable value, the value "negative infinity" is returned, which is designated as -Infinity. These special values denoting infinity behave exactly as one would expect: adding, subtracting, multiplying, or dividing infinity by any value results in an infinite (possibly with opposite sign).

Loss of significant digits occurs when the result of the arithmetic operation is closer to zero than the minimum possible value. In this case, the number returns 0. If the loss of significant digits occurs in a negative result, a special value, known as "negative zero," is returned.

Division by zero is not considered an error in JavaScript: in this case, simply returns infinity or negative infinity. However, there is one exception: the operation of dividing zero by zero does not have a well-defined value, so the special value "not-a-number", which is denoted as NaN, is returned as the result of such an operation. The NaN value is also returned when trying to divide infinity into infinity, extract the square root of a negative number, or perform an arithmetic operation with non-numeric operands that can’t be converted to numbers.

Text

A string is an unchanged, ordered sequence of 16-bit values, each of which usually represents a sign of Unicode.

The strings in JavaScript are the data type used to represent the text. The length of a string is the number of 16-bit values contained in it. The numbering of characters in strings (and elements in arrays) in the JavaScript language starts from zero: the first 16-bit value is in position 0, the second in position 1, etc.

An empty string is a string whose length is 0. In JavaScript, there is no special type for representing a single element of a string. To represent a single 16-bit value, simply use a string with a length of 1.

To include a literal string in a JavaScript program, it's enough just to enclose the string's characters in paired single or double quotes ('or'). Double-quoted characters can be contained in strings delimited by single-quote characters, and single-quote characters in strings delimited by double-quote characters.

The backslash (\) has a special purpose in JavaScript strings. Along with the characters following it, it denotes a character that is not representable within the string in other ways. For example, \ n is an escape sequence that denotes a line feed character.

Another example is the sequence \', which stands for the single-quote character.

Work with strings

One of the built-in JavaScript features is the ability to concatenate strings. If the + operator is applied to numbers, they are added up, and if to the rows - they are combined, with the second line added to the end of the first.

For example:

```
msg = "Hello," + "world"; // Get the string "Hello, world"

greeting = "Welcome to my homepage," + "" + name;
```

To determine the length of a string-the number of 16-bit values contained in it-the length property is used. For example, the length of the string s can be obtained as follows:

```
s.length
```

Besides, in addition to the length property, strings have many methods (see the reference section for more information):

```
var s = "hello, world" // Let's start with the same text.

s.charAt (0) // => "h": the first character.

s.charAt (s.length-1) // => "d": last character.

s.substring (1,4) // => "ell": the 2nd, 3rd and 4th characters.

s.slice (1,4) // => "ell": the same

s.slice (-3) // => "rld": last 3 characters

s.indexOf ("l") // => 2: position of the first character l.

s.lastIndexOf ("l") // => 10: position of the last character l.

s.indexOf ("l", 3) // => 3: position of the first character "l", the next

                   // for 3 characters in a line

s.split (",") // => ["hello", "world"] breaks into substrings

s.replace ("h", "H") // => "Hello, world": replaces all occurrences of substring

s.toUpperCase () // => "HELLO, WORLD"
```

Date and time

In the base JavaScript language, there is a Date () constructor to create objects that represent the date and time. These Date objects have methods for performing simple calculations involving dates. The Date object is not a fundamental data type, like numbers. A full description can be found in the reference section.

```
var then = new Date (2010, 0, 1); // First day of the first month of 2010
var later = new Date (2010, 0, 1, 17, 10, 30);        // The same date, at 17:10:30
                                                      // local time
var now = new Date (); // Current date and time
var elapsed = now - then; // Difference of dates: interval in milliseconds
later.getFullYear () // => 2010
later.getMonth () // => 0: months count starts from zero
later.getDate () // => 1: The days count starts with a unit
later.getDay () // day of the week. 0 - resurrection, 5 - spots.
later.getHours () // 17 hours local time
later.getUTCHours () // hours by UTC; depends on the time zone
later.toString () // "Fri Jan 01 2010 17:10:30 GMT + 0300"
later.toUTCString () // "Fri, 01 Jan. 2010 14:10:30 GMT"
later.toLocaleDateString () // "January 1, 2010"
later.toLocaleTimeString () // "17:10:30"
```

Null and undefined

The null keyword in JavaScript language has a special purpose and is usually used to indicate the absence of a value. The typeof operator for null returns the string "object", which means that null is a special "empty" object. However, in practice, the value of null is usually considered to be the only member of a proprietary type and can be used as a sign of no value, such as a number, string, or object.

In the JavaScript language, there is one more value indicating that there is no value. The value is undefined, indicating the complete absence of any value. It returns when you access a variable that has never been assigned a value, as well as a nonexistent property of the object or array element. In addition, the value of undefined is returned by functions that do not have a return value, and is assigned to function parameters for arguments that were not passed on the call.

Boolean values

A logical value indicates the truth or falsity of something. A logical data type has only two valid logical values. These two values are represented by literals true and false.

Boolean values are usually the result of comparison operations performed in JavaScript programs.

For example:

```
a = = 4
```

This expression checks whether the value of a is equal to the value of 4. If so, the result of this comparison is a logical value of true. If the value of a is not 4, the result of the comparison is false.

Boolean values are commonly used in JavaScript control constructs. For example, the if / else statement in JavaScript performs one action if the Boolean value is true, and another action if false. Typically, a comparison that creates a Boolean value is directly combined with the instruction in which it is used. The result looks like this:

```
if (a = = 4)
    b = b + 1;
else
    a = a + 1;
```

Equality and inequality operators

The = = and = = = operators check two values for a match, using two different match definitions. Both operators accept operands of any type and return true if their operands match, and false if they are different. The = = = operator, known as an identity operator, checks two operands for "identity", guided by a strict definition of a match. The = = operator, the equality operator, checks whether its two operands are equal in accordance with a less strict definition of match that allows type conversions.

In the JavaScript language, the operators =, = = and = = = are supported. Make sure that you understand the difference between the assignment, equality and identity operators.

Be careful and apply the right operators when developing your programs! It is very tempting to call all three operators "equal", but in order to avoid confusion it is better to read the statement = how "it turns out" or "assigned", the operator = = read as "equal", and the word "identically" means the operator = = =.

Operators != And ! = = mean "not equal" and "not identical".

The identity operator = = = computes the values of its operands, and then compares the two values, without converting the types, according to the following rules:

1. If two values have different types, they are not identical.
2. If both operands are null or undefined, they are identical.

3. If both operands are a Boolean value true or both are Boolean false, they are identical.

4. If one or both of the values are NaN, they are not identical. The value of NaN is never identical to any value, even to yourself! To check whether the value of x is NaN, use the expression x! = = x. The value of NaN is the only one for which such a test will return true.

5. If both values are numbers with the same value, they are identical. If one operand is 0 and the other is -0, they are also identical.

6. If both values are strings and contain the same 16-bit values in the same positions, they are identical. If the lines differ in length or content, they are not identical. Two lines can have the same meaning and look the same on the screen, but contain different sequences of 16-bit values. The JavaScript interpreter does not normalize Unicode characters, so these pairs of strings are not considered equal = = = and = = neither equal nor identical.

7. If both values refer to the same object, array, or function, then they are identical. If they refer to different objects (arrays or functions), they are not identical, even if both objects have identical properties.

The equality operator = = is similar to the identity operator, but it uses less stringent rules. If the values of the operands are of different types, it performs type conversion and tries to perform a comparison:

1. If two values are of the same type, they are checked for identity, as described above. If the values are identical, they are equal; if they are not identical, they are not equal.

2. If two values do not refer to the same type, the = = operator can still consider them equal. The following rules and type conversions are used:

3. If one value is null and the other is undefined, then they are equal.

4. If one value is a number and the other is a string, the string is converted to a number and a comparison is made with the converted value.

5. If any value is true, it is converted to 1 and the comparison is executed again. If any value is false, it is converted to 0 and the comparison is executed again.

6. If one of the values is an object and the other is a number or string, the object is converted to a simple type and the comparison is executed again. The object is converted to a value of a simple type either using its toString () method, or using its valueOf () method. The built-in JavaScript base class classes first try to perform the valueOf () conversion, and then toString (), except for the Date class, which always performs the toString () conversion. Objects that are not part of the basic JavaScript can transform themselves into values of simple types in the way defined by their implementation.

7. Any other combination of values is not equal.

As an example of equality testing, consider the comparison:

"1" = = true

The result of this expression is true, i.e. these different looking values are actually equal. The boolean value true is converted to the number 1, and the comparison is executed again. Then, the string "1" is converted to the number 1. Since both numbers now match, the comparison operator returns true.

Practical Work. The simplest calculations

Write a program that tests the formulas of abridged multiplication.

For all formulas, it is necessary to program the calculation of the right and left parts of the formula.

Add the necessary code to the next program.

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>John Smith</title>

<script>

function HTML_Container(elem){

  this.elem = elem;

  this.out = function(s){

    this.elem.innerHTML = s;

  }

  this.clr = function(){

    this.elem.innerHTML = "";

  }

  this.add = function(s){

    this.elem.innerHTML += s;

  }

}
```

```
</script>
</head>
<body>
<div id="my">conteiner</font></div>
<script>
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
cnt.out("<font color='#0000FF' size='+3'>The simplest calculations
</font><br>");
a = 2;
b = 7;
cnt.add("a = "+a+"<br>");
cnt.add("b = "+b+"<br>");
c1 = (a+b)*(a+b);
c2 = a*a + 2*a*b + b*b;
cnt.add("(a + b)^2= " + c1+"<br>");
cnt.add("a^2 + 2ab + b^2 = " + c2+"<br>");
cnt.add("<br><h1><i> Add code for other sections </i></h1><br>");
</script>
</body>
</html>
```

Formulas for squares

$(a + b)^2 = a^2 + 2ab + b^2$

$(a – b)^2 = a^2 – 2ab + b^2$

$a^2 – b^2 = (a – b)(a + b)$

$(a + b + c)^2 = a^2 + b^2 + c^2 + 2ab + 2ac + 2bc$

Formulas for cubes

$(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$

$(a – b)^3 = a^3 – 3a^2b + 3ab^2 – b^3$

$a^3 + b^3 = (a + b)(a^2 – ab + b^2)$

$a^3 – b^3 = (a – b)(a^2 + ab + b^2)$

Formulas for the fourth degree

$(a + b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$

$(a – b)^4 = a^4 – 4a^3b + 6a^2b^2 – 4ab^3 + b^4$

$a^4 – b^4 = (a – b)(a + b)(a^2 + b^2)$

Practical work. Trigonometric formulas. Module Math

Write a program that tests the formulas of abridged multiplication.

For all formulas, it is necessary to program the calculation of the right and left parts of the formula (if the part is not equal to a constant). Add the code to the next HTML page.

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title> John Smith</title>

<script>

function HTML_Container(elem){

  this.elem = elem;

  this.out = function(s){

    this.elem.innerHTML = s;

  }

  this.clr = function(){
```

```
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
</script>
</head>
<body>
<div id="my">conteiner</div>
<script>
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
cnt.out("<font color='#0000FF' size='+3'> Trigonometric formulas.
Module Math.</font><br>");
gr_alfa = 67;
gr_beta = 11;
alfa = Math.PI/180*gr_alfa;
beta = Math.PI/180*gr_beta;
cnt.add("<b>Исходные данные</b><br>");
cnt.add("gr_alfa = "+gr_alfa+"<br>");
cnt.add("gr_beta = "+gr_beta+"<br>");
cnt.add("rd_alfa = "+alfa+"<br>");
cnt.add("rd_beta = "+beta+"<br>");
cnt.add("<br><b> Basic Trigonometric Formulas </b><br>");
tmp = Math.sin(alfa)*Math.sin(alfa)+Math.cos(alfa)*Math.cos(alfa);
```

```
cnt.add("sin^2(α)+cos^2(α) =
Math.sin(alfa)*Math.sin(alfa)+Math.cos(alfa)*Math.cos(alfa) =
"+tmp+"<br>");

tmp = 1+Math.tan(alfa)*Math.tan(alfa);

cnt.add("1+tg^2(α) = 1+Math.tan(alfa)*Math.tan(alfa) =
"+tmp+"<br>");

tmp = 1/Math.cos(alfa)/Math.cos(alfa);

cnt.add("1/cos^2(α) = 1/Math.cos(alfa)/Math.cos(alfa) =
"+tmp+"<br>");

cnt.add("<i> Basic Trigonometric Formulas </i><br>");

cnt.add("<br><h1><i> Add code for other sections </i></h1><br>");

</script>

</body>

</html>
```

Practical work. Logical Operations

Write a program that creates the following objects:
HTML_Container – output information to the container.
FORM_data – reading information from the form.
Triangle - a triangle, in this object should be the following elements:

- sides a, b, c;
- property tp - possible values: rectangular, obtuse, acute-angled, does not exist.
- methods init_tri, def_tp, toString (see the program below).

When you click the "compute" button, an instance of the "triangle" object is created and information about it is displayed (see the program below).

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title> John Smith </title>
```

```
<script>
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
}
function Triangle(a, b, c) {
  this.a = Number(a);
  this.b = Number(b);
  this.c = Number(c);
  this.init_tri = function(){
    alert("sort the sides of the triangle, with should be the
largest ");
```

```
  }
  this.def_tp = function () {
    return " Return value rectangular, obtuse, acute, does not
exist ";
  }
  this.init_tri();
  this.tp = this.def_tp();
  this.toString = function () {
    return "Triangle : " + this.a + " " + this.b + " " + this.c +
" (" + this.tp + ")";
  }
}
</script>
</head>
<body>
<script>
function myCompute(){
    var elem = document.getElementById('my');
    var cnt = new HTML_Container(elem);
    var myf = new FORM_data("myForm");
    var a = myf.get("a");
    var b = myf.get("b");
    var c = myf.get("c");
    var tri = new Triangle(a, b, c);
    cnt.out("<font color='#0000FF'
size='+3'>triangle</font><br>");
```

```
  cnt.add(tri+"<br>");
}
</script>
<form name="myForm">
a: <input name="a" type="text" value="5"><br>
b: <input name="b" type="text" value="4"><br>
c: <input name="c" type="text" value="3"><br>
<input name="compute" type="button" onClick="myCompute()"
value="compute">
</form>
<div id="my">conteiner</div>
</body>
</html>
```

Practical Work. Switch Statement

Write a program that translates a number from one number system to another.

The program creates the following objects:

HTML_Container – output information to the container.

FORM_data – reading information from the form.

In the "Source data" field the initial number is written, the result is output to the container DIV.

The choice of the initial and final number system is carried out using the drop-down menu.

It is necessary to provide the following translation of numbers:

"2->8"

"2->10"

"2->16"

"8->2"

"8->10"

"8->16"

"10->2"

"10->8"

"10->16"

"16->2"

"16->8"

"16->10"

Pressing the "compute" button translates the number from one system to another and displays the result (see the program below).

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<style>

  .myclass {

    background-color: #CCFFFF;

    color: #CC0000;

    border: solid;

    border-color: #0000FF;

    width: 300px;

    height: auto;

}

</style>

<title> John Smith </title>

<script>

function HTML_Container(elem){

  this.elem = elem;
```

```
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
}
</script>
</head>
<body>
<script>
function myCompute(){
  var elem = document.getElementById('my');
  var cnt = new HTML_Container(elem);
  var myf = new FORM_data("myForm");
  var act = myf.get("from_to");
```

```
  var src = myf.get("src");
  var res = 999999;
  var rout = "Error";
  switch (act) {
  case "2-8":
    res = parseInt(src,2)
    rout = res.toString(8);
    break;
  case "16-10":
    res = parseInt(src,16)
    rout = res.toString(10);
    break;
  default:
    alert( 'Error' );
}
  cnt.out(rout);
}
</script>
<form name="myForm">
Initial data:<input name="src" type="text" value="111001101">
// Add more translation options.
<select name="from_to">
  <option value="2-8">2-&gt;8</option>
  <option value="16-10">16-&gt;10</option>
</select>
```

```
<input name="compute" type="button" onClick="myCompute()"
value="compute">

</form>

Результат: <div class="myclass" id="my">conteiner</div>

</body>

</html>
```

Practical Work. Strings. Keyboard Layout

Write a program that translates a string from one keyboard layout to another.

The program creates the following objects:

– FORM_data – reading information from form fields and writing information to form fields.

– SuperStr – methods of this object allow you to translate from one keyboard layout to another.

1. Create a page index.html.

– create a text field "Russian",
– create a text field "English",
– create a button "Rus-> Lat",
– create a button "Lat-> Rus",
– create a button "Clear".

2. The algorithm of operation.

When you press the "Rus-> Lat" button, the line from the "Russian" field is translated into the Latin layout (for example: "йцукен123" -> "qwerty123") and is written in the "English" field.

When you click the "Lat-> Rus" button, the line from the "English" field is translated into the Russian layout (for example: "qwerty123" -> "йцукен123") and is written in the field "Russian".

When the "Clear" button is pressed, the "English" and "Russian" fields are cleared (they contain blank lines).

It is necessary to supplement the code in the next program.

```
<html>

<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>John Smith</title>
<script>
// read and write data form
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}
// SuperStr constructor
function SuperStr(){
  var symRUS =
"йцукенгшщзхъфывапролджэячсмитьбюЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮ"
;
  var symLAT =
"qwertyuiop[]asdfghjkl;'zxcvbnm,.QWERTYUIOP{}ASDFGHJKL:\"ZXCVBNM<>
";
  this.RusLat = function(strIn){
    var s = "", i, ch, n;
    for(i=0; i < strIn.length; i++){
    ch = strIn.charAt(i);
    n = symRUS.lastIndexOf(ch);
```

```
    if(n != -1) ch = symLAT.charAt(n);
    s += ch;
    }
    return s;
  }
  this.LatRus = function(strIn){
    alert("Create a translation method from the Latin keyboard
layout into Russian");
  }
}
</script>
</head>
<body>
<script>
// functions
function rus_lat(){
  var s0 = myf.get("inpRU");
  var s1 = sups.RusLat(s0);
  myf.set("inpEN",s1);
}
function lat_rus(){
  alert("Create a procedure for pressing the button Lat -> Rus ");
}
function clr_fld(){
  alert("Create a procedure for clearing input fields ");
}
```

```
</script>

<form name="myForm">

English: <input name="inpEN" type="text" value=""><br />

Russian: <input name="inpRU" type="text" value=""><br />

<input name="LatRus" type="button" onClick="lat_rus()" value="Lat
-> Rus">

<input name="Clr" type="button" onClick="clr_fld()" value="Clear">

<input name="RusLat" type="button" onClick="rus_lat()" value="Rus
-> Lat">

</form>

<script>

// Globals

var myf = new FORM_data("myForm");

var sups = new SuperStr();

</script>

</body>

</html>
```

Practical Work. Strings and Arrays. Transliteration

In the SuperStr object (See "Strings. Keyboard Layout"), add a method that transliterates the Russian character string.

It is necessary to supplement the code in the next program. All the necessary actions are described using the alert function.

Compare this program with "Strings. Keyboard Layout" program and find significant differences.

```
<html>

<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style>
.myclass {
  background-color: #CCFFFF;
  color: #CC0000;
  border: solid;
  border-color: #0000FF;
  width: 300px;
  height: auto;
}
</style>
<title> John Smith</title>
<script>
// Global vars for SuperStr
var symRUS =
"йцукенгшщзхъфывапролджэячсмитьбюёЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮ
Ё";
var symLAT =
"qwertyuiop[]asdfghjkl;'zxcvbnm,.`QWERTYUIOP{}ASDFGHJKL:\"ZXCVBNM<
>~";
arrTRL = [];
 arrTRL[0] = "y";   arrTRL[33] = "Y";   //й
 arrTRL[1] = "ts";  arrTRL[34] = "TS";  //ц
                                        //у
                                        //к
                                        //е
```

```
//н
//г
//ш
//щ
//з
//х
//ъ
//ф
//ы
//в
//а
//п
//р
//о
//л
//д
//ж
//э
//я
//ч
//с
//м
//и
//т
//ь
```

```
                                        //б
arrTRL[31] = "yu";  arrTRL[64] = "YU";  //ю
arrTRL[32] = "yo";  arrTRL[65] = "YO";  //ё
alert("Fill all array arrTRL");

// End of global vars for SuperStr

// Write into container
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
// read and write data form
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
```

```
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}
// SuperStr constructor
function SuperStr(){
  this.RusLat = function(strIn){
    var s = "", i, ch, n;
    for(i=0; i < strIn.length; i++){
      ch = strIn.charAt(i);
      n = symRUS.lastIndexOf(ch);
      if(n != -1) ch = symLAT.charAt(n);
      s += ch;
    }
    return s;
  }
  this.LatRus = function(strIn){
    var s = "", i, ch, n;
    for(i=0; i < strIn.length; i++){
      ch = strIn.charAt(i);
      n = symLAT.lastIndexOf(ch);
      if(n != -1) ch = symRUS.charAt(n);
      s += ch;
    }
```

```
    return s;
  }
  this.Translit = function(strIn){
    alert("Add code of translateration function ");
  }
}
</script>
</head>
<body>
<script>
// functions
function rus_lat(){
  var s0 = myf.get("inpRU");
  var s1 = sups.RusLat(s0);
  myf.set("inpEN",s1);
}
function lat_rus(){
  var s0 = myf.get("inpEN");
  var s1 = sups.LatRus(s0);
  myf.set("inpRU",s1);
}
function translit(){
  alert("Add code of event handler ");
}
function clr_fld(){
```

```
  myf.set("inpRU","");
  myf.set("inpEN","");
  alert("Add cleanup code for source code and transliteration
result ");
}
</script>
<form name="myForm">
English: <input name="inpEN" type="text" value=""><br />
Russian: <input name="inpRU" type="text" value=""><br />
<input name="LatRus" type="button" onClick="lat_rus()" value="Lat
-> Rus">
<input name="Clr" type="button" onClick="clr_fld()" value="Clear">
<input name="RusLat" type="button" onClick="rus_lat()" value="Rus
-> Lat"><br />
<br />Russian text for translit:<br />
<textarea name="Trans" cols="20" rows="5"></textarea><br />
<input name="Translit" type="button" onClick="translit()"
value="Translit">
</form>
<hr />
Translit result: <div class="myclass" id="my">conteiner</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
var myf = new FORM_data("myForm");
var sups = new SuperStr();
```

```
</script>

</body>

</html>
```

Practical Work. Arrays. Simple Operations

The purpose of the work is to study and apply the methods of the Array object.

It is necessary to supplement the code in the next program. All the necessary actions are described using the alert function. As much as possible, use the methods of the Array object (rather than your own code).

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<style>

  .myclass {

    background-color: #CCFFFF;

    color: #CC0000;

    border: solid;

    border-color: #0000FF;

    width: 300px;

    height: auto;

}

</style>

<title> John Smith </title>

<script>


// Constructor of output container object
```

```
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}

// Constructor of read and write data form object
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}
</script>
</head>
```

```
<body>
<script>
// functions
function process(){
  var s0 = myf.get("src_data");
  var arr = s0.split(" ");

  cnt.out("<b> Initial data</b><br>");
  cnt.add(arr+"<br>");

  cnt.add("<b> Number of entered numbers </b><br>");
  alert("Output to the container the number of entered numbers ");

  cnt.add("Sort Ascending </b><br>");
  var tmp = [];
  alert("Display the result of sorting in ascending order ");

  cnt.add("<b> Sorting in decreasing order </b><br>");
  alert("Display the result of sorting in descending order ");

  cnt.add("<b>Minimum </b><br>");
  alert("Output to the container the minimum element ");

  cnt.add("<b> Maximum </b><br>");
  alert("Display the maximum element in the container ");
```

```
  cnt.add("<b> Sum of elements </b><br>");
  alert("Display the sum of the elements in the container ");

  cnt.add("<b> Average value </b><br>");
  alert("Output to the container the average value of the elements
");

}
function clr_fld(){
  alert("Clear input and output fields ");
}
</script>
<form name="myForm">
Array elements, separated by spaces:<br />
<textarea name="src_data" cols="40" rows="5">10 9 7 8 5 6 1 2 3
4</textarea><br />
<input name="Process" type="button" onClick="process()"
value="Data processing">
<input name="Clr" type="button" onClick="clr_fld()" value="Clear">
</form>
<hr />
Result: <div class="myclass" id="my">conteiner</div>
<script>
// Globals
var elem = document.getElementById('my');
```

```
var cnt = new HTML_Container(elem);

var myf = new FORM_data("myForm");

</script>

</body>

</html>
```

Practical Work. Arrays and Objects

The purpose of the work is to study and apply in practice the compilation and use of complex hierarchical structures. Structures are compiled using a combination of objects and arrays.

Write a program for calculating and outputting your estimates for the course "Scripting programming languages", for this you need to supplement the code in the following program.

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title> John Smith</title>

<style>

td{

  border-collapse: collapse;

  border-color: #999999;

  border-style: solid;

  border-width: 1px;

}

table{

  border-collapse: collapse;
```

```
  border-color: #999999;
  border-style: solid;
  border-width: 1px;
}
.css_total {
  background-color: #CCFFFF;
  color: #CC0000;
  font-weight: bold;
}
.css_modul{
  background-color: #DBDBDB;
}
.css_task{
  background-color: #F9FFF9;
}
</style>
<script>
//Global array
var MyTasks =
  {nam:" Final Grade", cls:"css_total", mgr:100, rgr:0,
    mdl:[
          {nam:"Mod №1", cls:"css_modul", mgr:35, rgr:0,
            tsk:[
                  {nam:"LW №1.1", cls:"css_task", mgr:5, rgr:0},
                  {nam:"LW №1.2", cls:"css_task", mgr:5, rgr:0}
```

```
            ]
        },
        {nam:"Mod №2",  cls:"css_modul", mgr:35, rgr:0,
          tsk:[
                {nam:"LW №2.1", cls:"css_task", mgr:5, rgr:0},
                {nam:"LW №2.2", cls:"css_task", mgr:5, rgr:0}
            ]
        },
        {nam:" Exam ", cls:"css_modul", mgr:30, rgr:0,
          tsk:[
                {nam:"Ind №1", cls:"css_task", mgr:5, rgr:0},
                {nam:"Ind №2", cls:"css_task", mgr:5, rgr:0}
            ]
        }
      ]
  }
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
```

```
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}

// Constructor of read and write data form object
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
}
</script>
</head>
<body>
<script>
// functions
function compute_marks(){
  var i,j,n,k,rgr,mgr;
  var s0 = myf.get("src_data");
  var arr = s0.split(" ");
  n=0;
```

```
  for(i=0; i<MyTasks.mdl.length; i++){
    rgr = 0;
    mgr = 0;
    for(j=0; j< MyTasks.mdl[i].tsk.length; j++){
      k=0;
      if(n<arr.length) k=Number(arr[n++]);
      MyTasks.mdl[i].tsk[j].rgr = k;
      rgr += k;
      mgr += MyTasks.mdl[i].tsk[j].mgr;
    }
    MyTasks.mdl[i].rgr=Math.round(rgr*MyTasks.mdl[i].mgr/mgr);
    // MyTasks.mdl[i].rgr=rgr*MyTasks.mdl[i].mgr/mgr;
    MyTasks.rgr += MyTasks.mdl[i].rgr;
  }
}

function out_marks(){
  var i,j,s;
  cnt.clr();
  s = "<table border=1>";
  s += "<tr class='"+MyTasks.cls+"'>";
  s += "<td>"+MyTasks.nam+"</td>";
  s += "<td>"+MyTasks.mgr+"</td>";
  s += "<td>"+MyTasks.rgr+"</td>";
  s += "</tr>";
```

```
  for(i=0; i<MyTasks.mdl.length; i++){
    s += "<tr class='"+MyTasks.mdl[i].cls+"'>";
    s += "<td>"+MyTasks.mdl[i].nam+"</td>";
    s += "<td>"+MyTasks.mdl[i].mgr+"</td>";
    s += "<td>"+MyTasks.mdl[i].rgr+"</td>";
    s += "</tr>";
    for(j=0; j< MyTasks.mdl[i].tsk.length; j++){
      s += "<tr class='"+MyTasks.mdl[i].tsk[j].cls+"'>";
      s += "<td>"+MyTasks.mdl[i].tsk[j].nam+"</td>";
      s += "<td>"+MyTasks.mdl[i].tsk[j].mgr+"</td>";
      s += "<td>"+MyTasks.mdl[i].tsk[j].rgr+"</td>";
      s += "</tr>";
    }
  }
  s += "</table>";
  cnt.add(s);
}

function process(){
  compute_marks();
  out_marks();
}

function clr_marks(){
  var i,j;
```

```
  for(i=0; i<MyTasks.mdl.length; i++){
    for(j=0; j< MyTasks.mdl[i].tsk.length; j++){
      MyTasks.mdl[i].tsk[j].rgr = 0;
    }
    MyTasks.mdl[i].rgr= 0;;
  }
  MyTasks.rgr = 0;
  out_marks();
}
</script>
<form name="myForm">
All my marks, separated by spaces:<br />
<textarea name="src_data" cols="40" rows="5">0 1 2 3 4 5 0 1 2 3 4
5</textarea><br />
<input name="Process" type="button" onClick="process()"
value="Data processing">
<input name="Process" type="button" onClick="clr_marks()"
value="Clear marks">
</form>
<hr />
Result: <div class="myclass" id="my">conteiner</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
var myf = new FORM_data("myForm");
```

```
</script>

</body>

</html>
```

Self-examination questions

1. The side of the square is given. Find its perimeter.
2. The radius of the circle is given. Find its diameter.
3. Considering that the Earth is an ideal sphere with a radius of R = 6350 km, determine the distance to the horizon line from a point with a given height above the Earth.
4. The length of the edge of the cube is given. Find the volume of the cube and the area of its lateral surface.
5. The radius of the circle is given. Find the circumference and the area of the circle.
6. Determine the maximum and minimum values from two different real numbers.
7. Two distances are known: one in kilometers, the other in feet (1 foot = 0.45 m). Which of the distances is smaller?
8. Two speeds are known: one in kilometers per hour, the other in meters per second. Which speed is greater?
9. The radius of the circle and the side of the square are given. Which figure has an area larger?
10. The volumes and masses of two bodies from different materials are given. Which body material has a high density?
11. There are known resistances of two unconnected parts of the electrical circuit and voltage on each of them. On what area is the current flowing?
12. Real numbers are given a,b,c ($a \neq 0$). Find out if the equation ax2+bx+c =0 has real roots.
13. Write a program which, depending on the serial number of the day of the week (1,2, ..., 7) displays the name (Monday, Tuesday, ..., Sunday).
14. Create a program, which, depending on the serial number of the month (1,2, ..., 12) displays its name (January, February, ..., December)
15. Create a program, which, depending on the sequence number of the month (1,2, ..., 12) displays the time of the year to which this month applies.
16. The following serial numbers are conventionally assigned to playing card games: suits "peaks" - 1, suits "clubs" - 2, suits "tambourines" 3, suits "hearts" - 4. Under the chosen suit number m($1 \leq m \leq 4$), determine the name of the corresponding suit.
17. The following serial numbers are conditionally assigned to playing cards, depending on their dignity: "jack" - 11, "lady" - 12, "king" - 13, "ace" - 14. Orders of

the remaining cards correspond to their names ("six", " nine ", etc.). With the given card number k($6 \le k \le 14$), determine the value of the corresponding card.

18. Make the program: a) which requests the name of the person and repeats it on the screen; b) which requests the name of the person and repeats it on the screen with a greeting.

19. Make a program that requests the name of the football team and repeats it on the screen with the words "- this is the champion!"

20. Create a program that requests a separate name and a separate surname, and then displays them as one character string.

21. Make a program that asks for the name of the country and its capital, and then displays the message: "The capital of the state ... is the city ..." (the corresponding values should be displayed in place of the dots).

22. Make a program that asks for the title of the novel and the surname of its author, and then displays the message: "Writer ... is the author of the novel ..." (the corresponding values should be displayed in place of the dots).

23. The names of the two countries are given. Assign these names to the variables s1 and s2, then assign the name s2 to t1, and s1 to t2.

24. Given the name of the football club. Determine the number of characters in it.

25. The name of the city is given. Determine whether or not the number of characters in it is even.

26. Two names are given. Determine which one is longer.

27. The names of the three cities are given. Display the longest and the shortest name.

28. The names of the two countries are given. Assign these names to the variables s1 and s2, and then exchange values for s1 and s2.

29. To create a program for exchanging the values of three variable a, b, c with a string type according to the following scheme: a) b assign the value of c, a assign the value of b, c assign the value a; b) b assign the value of a, c assign the value of b, a assign the value c.

30. Given the name of the football club. Print it on the screen with a "bar".

31. Create a program that prints the specified word starting with the last letter.

32. Given the word s1. Get the word s2, formed by the odd letters of the word s1.

33. Given the word s. Get the word t, obtained by reading the word s from its end.

34. Get a string consisting of five stars ("*").

35. Get a string consisting of eight "_" characters.

36. Make a program that forms a string consisting of any given number of any identical characters.

37. Given the word. Add to it four "+" symbols at the beginning and five "-" at the end.

38. Given the word. Add to him at the beginning and at the end as many stars as there are letters in this word.

39. Two words are given (the first is longer than the second). Replace in the second word the appropriate number of characters for the first word.

40. Given the proposal. Create a program that prints a "column" of all occurrences in the sentence of a character.

41. Fill the array: a) the first ten terms of the arithmetic progression with the known first term of the progression a and its difference p; b) the twenty first terms of the geometric progression with the known first term of the progression a and its denominator z; c) the twelfth first members of the Fibonacci sequence (sequences in which the first two terms are equal to 1, and the next two are equal to the sum of the two previous ones).

42. Identify: the sum of all elements of the array; the product of all elements of the array; the sum of the squares of all elements of the array; the sum of the first six elements of the array; the sum of the elements of the array from k1-st to k2-d (values k1 and k2 are entered from the keyboard; k2> k1); the arithmetic mean of all elements of the array; the arithmetic mean of all the elements of the array from s1-st to s2-d (values s1 and s2 are entered from the keyboard, s2> s1).

Part 2 Object Oriented Programming IN the JavaScript

Object Oriented Programming (OOP) means any kind of programming that uses a programming language with some object oriented constructs or programming in an environment where some object oriented principles are followed. At its heart, though, object oriented programming is a mindset which respects programming as a problem-solving dilemma on a grand scale which requires careful application of abstractions and subdividing problems into manageable pieces. Compared with procedural programming, a superficial examination of code written in both styles would reveal that object oriented code tends to be broken down into vast numbers of small pieces, with the hope that each piece will be trivially verifiable. OOP was one step towards the holy grail of software-re-usability, although no new term has gained widespread acceptance, which is why "OOP" is used to mean almost any modern programming distinct from systems programming, assembly programming, functional programming, or database programming. Modern programming would be better categorized as "multi-paradigm" programming, and that term is sometimes used. This book is primarily aimed at modern, multi-paradigm programming, which has classic object oriented programming as its immediate predecessor and strongest influence.

Historically, "OOP" has been one of the most influential developments in computer programming, gaining widespread use in the mid 1980s. Originally heralded for its facility for managing complexity in ever-growing software systems, OOP quickly developed its own set of difficulties. Fortunately, the ever evolving programming landscape gave us "interface" programming, design patterns, generic programming, and other improvements paving the way for more contemporary Multi-Paradigm programming. While some people will debate endlessly about whether or not a certain language implements "Pure" OOP – and bless or denounce a language accordingly – this book is not intended as an academic treatise on object oriented programming or its theory.

Instead, we aim for something more pragmatic: we start with basic OO theory and then delve into a handful of real-world languages to examine how they support OO programming. Since we obviously cannot teach each language, the point is to illustrate the trade-offs inherent in different approaches to OOP.

JavaScript is a high-level, dynamic, untyped and interpreted programming language that is well suited for programming in object-oriented and functional styles. We will consider some specific features of programming in JavaScript on the example of the object "canvas".

Methods and Properties of the Canvas Element

The canvas element appeared in HTML5. You can get the canvas object using the getContext () method

```
<canvas id="canvas" width="300" height="100"></canvas>
```

```
<script type="text/javascript">
var canvas = document.getElementById('canvas'); // get an element
var cnv = canvas.getContext('2d'); // get a canvas object
</script>
```

Now we can work with the cnv object.

Pen and Brush Style

The pen is for outputting outlines, and the brush is for pouring.

cnv.strokeStyle – property defines the pen style;

cnv.fillStyle – property defines the style of the brush.

The value of these properties can be:

A stream in the style of hexadecimal color specifiers. For example, "# ff0000";

RGB function, type "rgb (red, green, blue)". For example, "rgb (255,0,0)" and "rgb (100%, 0%, 0%)";

RGBA function with alpha channel, "rgba (red, green, blue, alpha)". For example «rgba (255,0,0,0.5)»

An object created by one of the following methods:

createLinearGradient (x1, y1, x2, y2) - creates a linear gradient object;

createRadialGradient (x1, y1, r1, x2, y2, r2) - creates a radial gradient object;

createPattern (image, type) – fill pattern, type argument can take values:

repeat – repeat the pattern;

repeat-x – repeat the pattern with x;

repeat-y – repeat the pattern with y;

no-repeat - do not repeat the template.

```
cnv.strokeStyle = "#FF0000";
cnv.fillStyle = "rgba(255, 0, 0, 0.5)";
```

Also, except for transparency in the RGBA color value, you can define a global transparency value using the globalAlpha property. It takes a value from 0 to 1.0.

Contours

Methods for working with contours:

cnv.beginPath() – starts a complex contour;

cnv.closePath() – ends the complex path;

cnv.moveTo(x, y) – move the current pen coordinates to (x, y);

cnv.lineTo(x, y) – draw a line from the current coordinates to those specified in (x, y);

cnv.stroke() – traverse the contour with the stylus;

cnv.fill() – paint the outline with a brush;

cnv.quadraticCurveTo(cp1x, cp1y, x, y) – displays the second-order curve;

cnv.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y) – outputs the Bezier curve;

cnv.arc(x, y, radius, startAngle, endAngle, anticlockwise) – outputs the arc;

cnv.arcTo(x1, y1, x2, y2, radius) – adds an arc to the contour;

cnv.clip() – adds the region of the current path to the current region in which output is possible.

cnv.isPointInPath(x, y) – determines whether the point is inside the path, returns true or false.

Quadrilaterals

Methods for working with quadrangles:

cnv.rect(x, y, width, height) – displays a quadrilateral;

cnv.fillRect(x, y, width, height) – the filled quadrilateral;

cnv.strokeRect(x, y, width, height) – the contour of the quadrilateral;

cnv.clearRect(x, y, width, height) – cleaning the quad.

Under the quadrangles, it is understood here is not just a visible figure. For example, the clearRect method can be used as an eraser.

Line styles

Properties specifying style lines:

cnv.lineCap – line view.

square – square;

round – circle;

butt is the butt.

cnv.lineWidth – width of the line;

cnv.lineJoin – specifies how to connect the lines.

round – with rounding;

bevel – with bevel;

miter – with anti-aliasing, (default).

Text

Methods and properties for working with text:

cnv.fillText (text, x, y [, maxw]) – prints text to specified coordinates, with maximum width maxw;

cnv.strokeText (text, x, y [, maxw]) – prints the outline of the text to the specified coordinates, with the maximum width maxw;

cnv.measureText (text) – returns a TextMetrics object that determines the approximate width of the specified text. (TextMetrics.width).

cnv.font – css a string of font parameters (For example: italic 30px sans-serif);

cnv.textAlign – sets the alignment of the text ("start", "end", "left", "right", "center");

cnv.textBaseline – definition of the baseline ("top", "hanging", "middle", "alphabetic", "ideographic", "bottom");

Coordinate transformation

Methods for converting coordinates:

cnv.rotate (angle) – rotates clockwise by the specified angle in radians;

cnv.translate (x, y) – linear coordinate shift;

cnv.scale (x, y) – scaling;

cnv.transform (m11, m12, m21, m22, dx, dy) – multiplies the current transformation matrix with the specified matrix;

cnv.setTransform (m11, m12, m21, m22, dx, dy) – sets the new transformation matrix.

Images

To apply an image in the canvas element, you need to get it in the beginning. There are several ways to do this:

1. Use the DOM by means of getElementsByTagName or getElementById.
2. Create images in the script:

```
var img = new Image ();
img.src = "myimg.png";
```

3. Implement using data: url

```
var img = "data: image / gif; base64, ... image data ...";
```

Further images can be displayed on canvas:

cnv.drawImage (img, x, y) – displays images;

cnv.drawImage (img, x, y, width, height) – displays images with scaling in the specified sizes;

cnv.drawImage (img, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight) – displays the parts of the image with scaling in the specified sizes.

Raster Data

Methods for working with raster data:

cnv.createImageData (w, h) – creates raster data of size w * h. Returns the ImageData object;

cnv.getImageData (x, y, w, h) – gets the raster data of the canvas in the specified rectangle. Returns the ImageData object;

cnv.putImageData (ImageData idata, x, y) – displays raster data on the canvas to the specified coordinates;

cnv.putImageData (ImageData idata, x, y, sx, sy, sw, sh) – print a portion of the raster data to the canvas to the specified coordinates.

Preserving styles and transformation matrices

cnv.save () – save the state;

cnv.restore () – restore the state.

It is also useful to get a string of the format data: url, which can be transferred to the server and save the image to a file. For this, the cnv.toDataURL () method is used, it returns the string data: url encoded with base64. By default, the data contains information about the PNG image, but you can also present the image in a different format, for example cnv.toDataURL ("image / jpeg").

Practical Work. Object Oriented Programming

Based on the web page (see below), create the following object constructors:

1) polyline_figure (inheritor of PROTO_figure);
2) polyline_filled (inheritor of polyline_figure) is a closed polygon, colored inside;
3) circle_filled (inheritor of circle_figure) is a circle that is shaded inside;
4) arc_filled (inheritor arc_figure) – a segment of a circle, filled up inside.

Fill color is the constructor parameters.

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<style>

  .myclass {

    background-color: #CCFFFF;

    color: #CC0000;

    border: solid;

    border-color: #0000FF;

    width: 500;

  }

  canvas {

    border: 1px dashed black;

  }

</style>
```

```
<title> John Smith </title>
<script>
// Write into container
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}

// read and write data form
function FORM_data(frm_name){
  this.frm = document.forms[frm_name];
  this.get = function(nm){
    return this.frm.elements[nm].value;
  }
  this.set = function(nm,vl){
    this.frm.elements[nm].value = vl;
  }
```

```
}

//Prototype for all figures
function PROTO_figure(o,c,ss,lw){
  this._origen = o;
  this._cont = c;
  this._strokeSt = ss;
  this._lineWi = lw;
  this.SelfDrow = function(){
    this._cont.lineWidth = this._lineWi;
    this._cont.strokeStyle = this._strokeSt;
  }
}

// LINE. Inherit from PROTO_figure
function line_figure(o,c,ss,lw,p1){
  this._line_end = p1;
  PROTO_figure.apply(this, arguments);
  var parentSelfDrow = this.SelfDrow;
  this.SelfDrow = function(){
    parentSelfDrow.call(this);
    this._cont.beginPath();
    this._cont.moveTo(this._origen[0],this._origen[1]);
    this._cont.lineTo(this._line_end[0],this._line_end[1]);
    this._cont.stroke();
```

```
  }
}

// TRIANGLE. Inherit from PROTO_figure
function triangle_figure(o,c,ss,lw,par){
  this._tr_end = par;
  PROTO_figure.apply(this, arguments);
  var parentSelfDrow = this.SelfDrow;
  this.SelfDrow = function(){
    parentSelfDrow.call(this);
    this._cont.beginPath();
    this._cont.moveTo(this._origen[0],this._origen[1]);
    this._cont.lineTo(this._tr_end[0],this._tr_end[1]);
    this._cont.lineTo(this._tr_end[2],this._tr_end[3]);
    this._cont.closePath();
    this._cont.stroke();
  }
}

// CIRCLE. Inherit from PROTO_figure
function circle_figure(o,c,ss,lw,r){
  this._cr_rad = r;
  this._st_angl = 0;
  this._end_angl = 2*Math.PI;
  PROTO_figure.apply(this, arguments);
```

```
  var parentSelfDrow = this.SelfDrow;
  this.SelfDrow = function(){
    parentSelfDrow.call(this);
    this._cont.beginPath();

this._cont.arc(this._origen[0],this._origen[1],this._cr_rad,this._
st_angl,this._end_angl);
    this._cont.stroke();
  }
}

// ARC. Inherit from CIRCLE_figure
function arc_figure(o,c,ss,lw,r,an1,an2){
  circle_figure.apply(this, arguments);
  this._st_angl = an1;
  this._end_angl = an2;
}
</script>
</head>
<body>
<script>
// functions
function process(){
  var scmd = myf.get("gr_obj");
  scmd = "new "+scmd;
  cnt.add(scmd+"<br>");
```

```
  var fig = eval(scmd);
  fig.SelfDrow();
}

/*
line_figure([200,200],context,'#ADFF2F',20,[400,400])
triangle_figure([10,10],context,'#8B008B',5,[400,400,5,490])
circle_figure([250,250],context,'#FF0000',10,100)
arc_figure([250,250],context,'#FF0000',10,100,0,2)
*/
</script>
<form name="myForm">
Graphic object: <input name="gr_obj" type="text" size="80"
value="line_figure([200,200],context,'#ADFF2F',20,[400,400])"><br
/>
<input name="Process" type="button" onClick="process()"
value="Process">
</form>
<hr />
Report:
<div class="myclass" id="myDiv">result</div>
<br />
Graphic area:
<hr />
<canvas id="myCanvas" width="500" height="500"></canvas>
<script>
```

```
// Globals

var elem = document.getElementById("myDiv");

var cnt = new HTML_Container(elem);

var myf = new FORM_data("myForm");

var canvas = document.getElementById("myCanvas");

var context = canvas.getContext("2d");

cnt.clr();

</script>

</body>

</html>
```

Self-examination questions

1. Name the principles of the PLO and tell everyone about it.
2. Define the notion of "class".
3. What is the field / attribute of the class?
4. How correctly to organize access to fields of a class?
5. Give the definition of "constructor".
6. What is the difference between the default constructors, the copying and the constructor with the parameters?
7. What modifications of the access level do you know, tell us about each of them.
8. Tell us about the features of a class with a single private constructor.
9. What do the keywords "this", "super" say, where and how can they be used?
10. Give the definition of "method".
11. What is the method signature?
12. What methods are called overloaded?
13. Can static methods overload static?
14. Tell me about the redefinition of methods.
15. Can the method take a different number of parameters (variable-length arguments)?
16. Is it possible to narrow the access level / type of the return value when the method is overridden?
17. How can I access the overridden methods of the parent class?
18. What transformations are called descending and ascending?
19. What is the difference between overload and overload?
20. Where can I initialize static / non-static fields?

Part 3 Object Models: DOM, BOM and JS

By itself, the JavaScript language does not provide for working with the browser.

He does not know about HTML at all. But it allows you to easily expand yourself with new functions and objects.

The figure 1 schematically shows the structure, which is obtained if you look at a set of browser objects from the "bird's eye view".

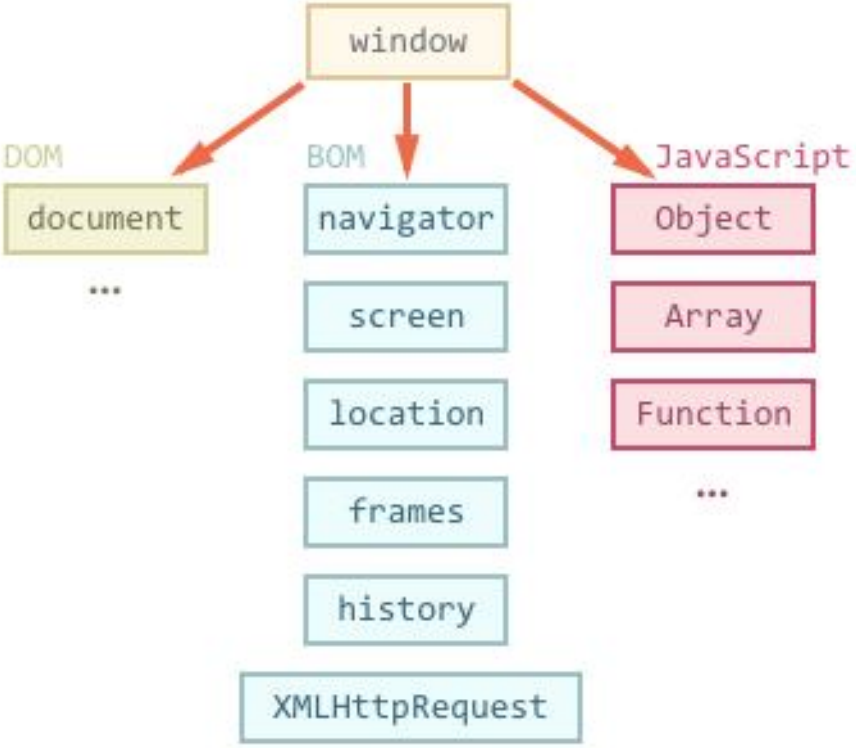

Figure 1 – Object models

As can be seen from the figure, the top is window.

This object has a dual position - on the one hand it is a global object in JavaScript, on the other hand it contains properties and methods for controlling the browser window, opening new windows, for example:

```
// open a new window / tab with the URL http://google.ru
window.open ('http://google.ru');
```

The Document Object Model (DOM)

The global document object allows you to interact with the contents of the page. Example of use:

```
document.body.style.background = 'red';
```

```
alert ('The BODY element has turned red, and now it's back');

document.body.style.background = '';
```

It and a huge number of its properties and methods are described in the standard W3C DOM.

For historical reasons, the first version of the DOM Level 1 standard once appeared, then came up with more properties and methods, and DOM Level 2 appeared, at the moment, DOM Level 3 is added on top of them and DOM 4 is being prepared.

Modern browsers also support some features that are not included in the standards, but de facto exist a long time ago and nobody wants to abandon them. They are conventionally called "DOM Level 0".

Also information on working with page elements can be found in the HTML 5 standard.

DOM Tree

The main tool of work and dynamic changes on the page is the DOM (Document Object Model) – the object model used for XML / HTML documents.

According to the DOM-model, the document is a hierarchy, a tree. Each HTML tag forms a tree node with an "element" type. Nested tags become child nodes. To represent the text, nodes with the type "text" are created.

The DOM is a document view in the form of an object tree, which is available for modification through JavaScript.

All that is in HTML, is in the DOM.

Even the <! DOCTYPE ...> directive we are putting at the beginning of HTML is also a DOM node, and is in the DOM tree just before <html>. Even the document object itself, formally, is the DOM node, the very root one.

There are 12 types of nodes in total, but in practice we work with four of them:

1. The document is the entry point to the DOM.
2. Elements – the main building blocks.
3. Text nodes – contain, in fact, the text.
4. Comments – sometimes they can include information that will not be shown, but is available from JS.

Autocorrection

When you read the wrong HTML, the browser automatically adjusts it for display and when building the DOM.

In particular, there will always be an upper <html> tag. Even if the text does not – in DOM it will, the browser will create it yourself.

The same goes for the <body> tag.

For example, if the file consists of one word "Hello", then the browser will automatically wrap it in <html> and <body>.

When generating the DOM, the browser itself handles errors in the document, closes the tags and so on.

Navigation on DOM-elements

DOM allows you to do anything with an HTML element and its contents, but you need to get the element you need first.

Access to the DOM begins with the document object. From it you can get to any knots.

The topmost tree elements are accessible directly from document.

<HTML> = document.documentElement

The first entry point is document.documentElement. This property refers to the DOM object for the <html> tag.

<BODY> = document.body

The second entry point is document.body, which corresponds to the <body> tag.

In modern browsers (except IE8-) there is also a document.head - a direct link to <head>

document.body can be null. You can not access an item that does not already exist when the script is executed.

In particular, if the script is in <head>, then document.body is not available in it.

Therefore, in the following example, the first alert displays null:

```
<!DOCTYPE HTML>

<html>

<head>

  <script>

    alert( "HEAD: " + document.body ); // null, body not yet

  </script>

</head>

<body>

  <script>

    alert( "BODY: " + document.body ); // body there is

  </script>
```

```
</body>

</html>
```

Here and further we will use two fundamentally different terms.
Child elements (or children) are the elements that lie directly within the given. For example, <HTML> usually contains <HEAD> and <BODY> inside.

Descendants - all the elements that lie within this, together with their children, the children of their children and so on. That is, the entire DOM subtree.
A pseudo-array (collection) **childNodes** stores all child elements, including text.

The example below will output the document.body children in series:

```
<!DOCTYPE HTML>

<html>

<body>

  <div>Stsrt</div>

  <ul>

    <li>Information</li>

  </ul>

  <div>End</div>

  <script>

    for (var i = 0; i < document.body.childNodes.length; i++) {

      alert( document.body.childNodes[i] ); // Text, DIV, Text,
UL, ..., SCRIPT

    }

  </script>

  ...

</body>

</html>
```

Let's pay attention to a small detail. If you run the example above, the last element is <script>. In fact, there is still a text in the document (indicated by a ellipsis), but at the time the script was executed, the browser did not reach it yet.

All navigation properties can only be used for read-only reading. You can not just replace an element with an assignment.

```
childNodes [i] = ....
```

The DOM is modified by other methods, which we will discuss later, all the navigation links are automatically updated.

The properties firstChild and lastChild provide quick access to the first and last element.

If there are child nodes, it is always true:

```
elem.childNodes[0] === elem.firstChild

elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
```

DOM-collections, such as childNodes and others, which we will see below, are not JavaScript arrays.

They do not have methods for arrays, such as forEach, map, push, pop, and others.

That is why childNodes are called "collection" or "pseudo-mass".

To enumerate the collection, you should use the usual for loop:

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

Access to the elements on the left and right of this can be obtained from the links previousSibling / nextSibling.

The parent is accessible via parentNode. If you go from one element to another for a long time, sooner or later you will reach the root of the DOM, that is, before the document.documentElement, and then the document.

The navigation links described above apply equally to all nodes in the document. In particular, in childNodes both text nodes and nodes-elements and nodes-comments coexist, if any.

But for most tasks, text nodes are not interesting to us.

So let's look at an additional set of links that do not take them into account:

These links are similar to those that used to be, only in a number of places is the word Element:

- children – only child nodes-elements, that is, corresponding to the tags.
- firstElementChild, lastElementChild – respectively, the first and last children-elements.
- previousElementSibling, nextElementSibling – Neighboring elements.
- parentElement is the parent element.

Equalities are always true:

```
elem.firstElementChild === elem.children[0]

elem.lastElementChild === elem.children[elem.children.length - 1]
```

Special references for tables

Specific DOM elements can have their own additional links for greater ease of navigation.

Here we will consider the table, since this is an important special case and just an example.

The list below shows the most useful:

TABLE

- table.rows – collection of rows of the TR table;
- table.caption / tHead / tFoot – references to elements of the table CAPTION, THEAD, TFOOT.
- table.tBodies is a collection of table TBODY elements, there may be several of them according to their specification.

THEAD / TFOOT / TBODY

- tbody.rows is a collection of strings of the TR section;
- tr.cells – collection of TD / TH cells;
- tr.sectionRowIndex – line number in the current section THEAD/TBODY;
- tr.rowIndex – row number in the table;
- td.cellIndex – the number of the cell in the line.

Example of use:

```
<table>

  <tr>

    <td>one</td> <td>two</td>

  </tr>

  <tr>

    <td>three</td>  <td>four</td>

  </tr>

</table>



<script>
```

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


alert( table.rows[0].cells[0].innerHTML ) // "one"

</script>
```

Additional search methods

If an element is assigned a special id attribute, you can get it directly from the variable with the name from the id value.

For example:

```
<div id="content-holder">

  <div id="content">Element</div>

</div>



<script>

  alert( content ); // DOM-element

  alert( window['content-holder'] ); // in the name is hyphen,

                                     // so through [...]

</script>
```

This behavior corresponds to the standard. It exists, first of all, for compatibility, as a fragment of the distant past and is not very welcome, because it uses global variables. The browser tries to help us by mixing the JS and DOM namespaces, but conflicts are possible.

A more correct and common practice is to access the element by calling document.getElementById ("identifier").

For example:

```
<div id="content"> Select this item </div>

<script>

  var elem = document.getElementById('content');

  elem.style.background = 'red';
```

```
  alert( elem == content ); // true

  content.style.background = ""; // same element

</script>
```

By the standard, the id value must be unique, that is, there can be only one element in the document with this id. And it is he who will be returned.

If the document has several elements with a unique id, then the behavior is undefined. That is, there is no guarantee that the browser will return the first or last - it will return randomly.

Therefore, try to follow the rule of uniqueness id.

The elem.getElementsByTagName(tag) method searches for all elements with the specified tag tag inside the elem element and returns them as a list.

The tag register does not matter.

Unlike getElementById, which exists only in the document context, the getElementsByTagName method can search within any element.

For example, we find all the input elements inside the table:

```
<table id="age-table">

  <tr>

    <td>Your age:</td>



    <td>

      <label>

        <input type="radio" name="age" value="young" checked>
under the age of 18      </label>

      <label>

        <input type="radio" name="age" value="mature"> from 18 to
50

      </label>

      <label>

        <input type="radio" name="age" value="senior"> over 60
</label>

    </td>
```

```
  </tr>


</table>


<script>
  var tableElem = document.getElementById('age-table');
  var elements = tableElem.getElementsByTagName('input');


  for (var i = 0; i < elements.length; i++) {
    var input = elements[i];
    alert( input.value + ': ' + input.checked );
  }
</script>
```

You can get all the children by passing the asterisk '*' instead of the tag:

```
// get all the elements of the document
document.getElementsByTagName ('*');
// get all descendants of elem:
elem.getElementsByTagName ('*');
```

One of the most common mistakes beginners (however, sometimes not only) is to forget the letter "s", that is, try to call the getElementByTagName method instead of getElementsByTagName.

The letter "s" is not needed where there is only one element, that is, getElementById, in other methods it is required.

The call to document.getElementsByName (name) allows you to get all the elements with this name attribute.

For example, all elements named age:

```
var elems = document.getElementsByName('age');
```

Until the appearance of the HTML5 standard, this method returned only those elements that support the attribute name, in particular: iframe, a, input, and others. In modern browsers, the tag does not matter.

This method is used very rarely.

The call elem.getElementsByClassName (className) returns a collection of elements with class className. Finds an element also in the event that it has several classes, and the desired one is one of them.

Supports all modern browsers. For example:

```
div class="article">Article</div>

<div class="long article">Long article</div>



<script>

  var articles = document.getElementsByClassName('article');

  alert( articles.length ); // 2, find both elements

</script>
```

Like getElementsByTagName, this method can be called both in the context of the DOM element and in the context of the document.

Practical Work. Window Object (part 1)

Edit the HTML page (see below) as follows.

When you click the "Open Windows" button, ALL pages with tasks from module 1 should be open.

When you click the "Close Windows" button, all open pages should be closed.

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<style>

  .myclass {

    background-color: #CCFFFF;

    color: #CC0000;

    border: solid;
```

```
    border-color: #0000FF;
}
</style>
<title>John Smith</title>
<script>
// window object list
var arrWIN = [];
// URLs list
var arrURL = [];
arrURL[0] =
"http://cdo.kname.edu.ua/mod/assignment/view.php?id=129211";
arrURL[1] =
"http://cdo.kname.edu.ua/mod/assignment/view.php?id=129212";
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
```

```
</script>
</head>
<body>
<script>
// functions
function openwin(){
  cnt.clr();
  for(var i=0; i<arrURL.length; i++){
    arrWIN[i] = window.open(arrURL[i]);
    cnt.add(i+": "+arrURL[i]+"<br>");
  }
}
function closewin(){
  for(var i=0; i<arrURL.length; i++){
    arrWIN[i].close();
  }
  cnt.out("No windows opened")
}
</script>
<form name="myForm">
<input name="open_win" type="button" onClick="openwin()"
value="Open windows">
<input name="close_win" type="button" onClick="closewin()"
value="Close Windows">
</form>
```

```
<hr />

Windows: <div class="myclass" id="my">No windows opened</div>

<script>

// Globals

var elem = document.getElementById('my');

var cnt = new HTML_Container(elem);

</script>

</body>

</html>
```

Practical Work. Window Object (part 2)

Change the HTML-pages (see below) as follows.

1. For each number, the main window (mainwin.html) should open the child window (subwin.html), which will perform the factorization of the number into 2 prime factors.

2. The number is passed as a parameter to the URL, for example, subwin.html? P = 998027.

3. The child window writes the result to its HTML_Container object in the following form:

- number1x number2;
- number1 must be less than 2;
- x is a small Latin letter.

For example, the number 998027 is the product of numbers 991 and 997. In the answer it is necessary to write 998027 = 991×997.

mainwin.html

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<style>

  .myclass {
```

```
    background-color: #CCFFFF;
    color: #CC0000;
    border: solid;
    border-color: #0000FF;
}
</style>
<title> John Smith (main window)</title>
<script>
// window object list
var arrWIN = [];
// URLs list
var arrURL = [];
arrURL[0] = "subwin.html?p=129211";
arrURL[1] = "subwin.html?p=129212";
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
```

```
  }
}


</script>
</head>
<body>
<script>
// functions
function openwin(){
  cnt.clr();
  for(var i=0; i<arrURL.length; i++){
    arrWIN[i] = window.open(arrURL[i]);
    cnt.add(i+": "+arrURL[i]+"<br>");
  }
}
function closewin(){
  for(var i=0; i<arrURL.length; i++){
    arrWIN[i].close();
  }
  cnt.out("No windows opened")
}
</script>
<form name="myForm">
<input name="open_win" type="button" onClick="openwin()"
value="Open windows">
```

```
<input name="close_win" type="button" onClick="closewin()"
value="Close Windows">
</form>
<hr />
Windows: <div class="myclass" id="my">No windows opened</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
</script>
</body>
</html>
```

subwin.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style>
  .myclass {
    background-color: #CCFFFF;
    color: #CC0000;
    border: solid;
    border-color: #0000FF;
}
</style>
```

```
<title> John Smith (child window)<</title>
<script>
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
</script>
</head>
<body>
Result: <div class="myclass" id="my">None</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
var param = location.search;
cnt.out(param);
</script>
```

```
</body>

</html>
```

Practical Work. Searching Items in the DOM

Part1

Change the HTML page index.html (see bellow) as follows.

Create a page index1.html (from index.html). When you click the button, you need to find all the "input" elements on the page and display the following information in the container:

The resulting collection (object).

For each item in the collection, print:

- collection item number;
- element itself (object);
- value of the element's value property;
- value of the element's name property;
- value of the cheked item property.

Part2

Change the HTML page index.html (see bellow) as follows.

Create a page index2.html (from index.html). When you click the button, you need to find all the elements with the names Res01 ... Res13 (names should be found in a loop, in the same loop search for a specific name) on the page and display all the elements of the collections received in the container.

For each collection, display the following information in the loop:

The resulting collection (object).

For each item in the collection, print:

- element number of the collection;
- element (object) itself;
- value of the element's value property;
- the value of the element's name property;
- the value of the cheked item property.

Part3

Change the HTML page index.html (see bellow) as follows.

Create a page index3.html (from index.html). When you click the button, you need to find all the elements with classes class_01 ... class_03 (names should be found in a loop, in the same loop search for a specific name) on the page and display all the elements of the collections received in the container.

For each collection, display the following information in the loop:
The resulting collection (object).
For each item in the collection, print:
1. Item number of the collection.
2. The very element (object).
3. List of children, for each child element:
- collection item number;
- element itself (object):
- value of the element's value property;
- value of the element's name property;
- value of the cheked item property.

index.html

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>John Smith</title>

<style>

.class_01 {

  background-color: #FFFFE0;

}

.class_02 {

  background-color: #FF0000;

}

.class_03 {

  background-color: #CCCCCC;

}

</style>



<script>
```

```
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
</script>
</head>
<body>
<script>
  function process(){
    alert("Add necessary actions")
  }
</script>
<form name="myForm">
<input name="Process" type="button" onClick="process()"
value="Process">
</form>
<hr />
```

```
Result: <div class="myclass" id="my">conteiner</div>
<hr />
<form name="dialForm">
<table width="100%" border="1" cellspacing="1" cellpadding="5">
  <tr>
    <th width="20" scope="col" bgcolor="#FFFFE0">OK</th>
    <th width="20" scope="col" bgcolor="#FF0000">Err</th>
    <th width="20" scope="col" bgcolor="#CCCCCC">Not</th>
    <th width="50" scope="col">Pic</th>
    <th scope="col">Text</th>
  </tr>
<tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res01" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res01" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res01" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im01.gif" /> </ td>
    <td valign = "top"> Multiple choice (closed question) with
only one correct answer (the student puts a mark in one of the
circles)
    </ td>
  </ tr>
  <tr>
```

```
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res02" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res02" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res02" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im01.gif" /> </ td>

    <td valign = "top"> Multiple choice (closed question) with one
or more correct answers (the student puts a mark in one or more
boxes)

  </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res03" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res03" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res03" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im03.gif" /> </ td>

    <td valign = "top"> Alternative question (True / False) </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res04" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res04" value = "OK" /> </ td>
```

```
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res04" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im04.gif" /> </ td>

    <td valign = "top"> Numeric question </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res05" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res05" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res05" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im05.gif" /> </ td>

    <td valign = "top"> Calculated question </ td>

  </ tr>

<tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res06" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res06" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res06" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im06.gif" /> </ td>

    <td valign = "top"> Nested Answers </ td>

  </ tr>
```

```
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input type = "radio" name = "Res07" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input type = "radio" name = "Res07" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input type = "radio" name = "Res07" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img / im07.gif" /> </ td>
    <td valign = "top"> The question of correspondence </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input type = "radio" name = "Res08" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input type = "radio" name = "Res08" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input type = "radio" name = "Res08" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img / im08.gif" /> </ td>
    <td valign = "top"> Short answer (open question) </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input type = "radio" name = "Res09" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input type = "radio" name = "Res09" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input type = "radio" name = "Res09" value = "OK" checked /> </ td>
```

```
    <td align = "center" valign = "top"> <img src = "ttv_img /
im09.gif" /> </ td>

    <td valign = "top"> Description </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res10" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res10" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res10" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im10.gif" /> </ td>

    <td valign = "top"> Essays </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res11" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res11" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res11" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im11.gif" /> </ td>

    <td valign = "top"> The goal in the image </ td>

  </ tr>

  <tr>
```

```
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res12" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res12" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res12" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im12.gif" /> </ td>

    <td valign = "top"> All or Nothing </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res13" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res13" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res13" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im13.gif" /> </ td>

    <td valign = "top"> On the correspondence (displacement) </
td>

  </ tr>

</table>

</form>

<script>

// Globals

var elem = document.getElementById('my');

var cnt = new HTML_Container(elem);
```

```
</script>

</body>

</html>
```

Self-examination questions

1. Is there any difference between windows and document?
2. Do windows.onLoad and document.onLoad call at the same time?
3. How to stop the further spread of the event?
4. Name the different paths to get the item from the DOM tree
5. Which method is best for getting an item through a CSS selector?
6. Can I delete the event handler for an item?
7. How to add a class to an element via a selector?
8. How can I check that one element is a child of another?
9. Which method is best for creating a DOM element?
10. How can I prevent multiple handler calls for one event?

Part 4 Browser Events

To respond to visitor actions and internal interaction of scripts, there are events.

An event is a signal from the browser that something has happened.

At the dawn of the World Wide Web, web developers had to deal with only a small number of events: "load", "click", "mouseover" and others. These fairly old types of events are well supported by all browsers. As the web platform evolved, more powerful application interfaces were included, and the number of events increased significantly. There is no standard that defines a complete set of events, and the number of supported events continues to increase rapidly. These new events are defined in the following two sources:

– Specification of "**DOM Level 3 Events**", which after years of stagnation has been actively developed under the auspices of the W3C consortium.

– Many new application interfaces in the **HTML5** specification (and related additional specifications) define new events used, for example, to manage the history of visits, the drag-and-drop mechanism (drag and drop), message exchange between documents, and audio and video playback videos.

Example of an event in an HTML tag.

```
<input name = "compute" type = "button" onClick = "myCompute ()"
value = "compute">
```

Event handler:

```
<script>

function myCompute(){

    var elem = document.getElementById('my');

    var cnt = new HTML_Container(elem);

    var myf = new FORM_data("myForm");

    var a = myf.get("a");

    var b = myf.get("b");

    var c = myf.get("c");

    var tri = new Triangle(a, b, c);

    cnt.out("<font color='#0000FF' size='+3'>tringle</font><br>");

//    cnt.add(tri.toString()+"<br>");
```

```
  cnt.add(tri+"<br>");

}

</script>
```

Document download events

Most web applications absolutely need a web browser to notify them when the document has finished loading and it will be ready to perform operations on it. The load event in the Window object serves this purpose. The load event is raised only after the document and all its images have been fully loaded. However, you can usually run scripts immediately after parsing the document before the images are loaded. You can significantly reduce the start time of the web application if you start executing scripts for events other than load.

The **DOMContentLoaded** event is fired as soon as the document is loaded, parsed by the parser, and all the pending scripts are executed. By this time, images and scripts with the async attribute can continue to load, but the document itself will already be ready for operations. This event was first introduced in Firefox and was subsequently borrowed by all other browser manufacturers, including Microsoft Corporation, which added support for this event in IE9. Despite the DOM prefix in the name, this event is not part of the standard of the DOM Level 3 Events event model, but it is standardized by the HTML5 specification.

Example of the load event.

```
window.onload = function(){

    var i,j,s;

    for(j=1; j<=13; j++){

      s="";

      if(j<10) s="0";

      s="Res"+s+j;

      ttvArrRadio[j-1] = document.getElementsByName(s);

      for (i = 0; i < ttvArrRadio[j-1].length; i++) {

        ttvArrRadio[j-1][i].addEventListener("change",ttvChange);

      }

    }
```

```
var btn = document.getElementById("res");

btn.addEventListener("click",ttvReset);}
```

Mouse events

- click – occurs when the item is clicked on with the left mouse button
- contextmenu – occurs when you right-click on an item
- mouseover – occurs when an element is hovered over the mouse
- mousedown and mouseup – when the mouse button is pressed or pressed
- mousemove – when moving the mouse

Events on the controls

- submit – the visitor sent the form <form>
- focus – the visitor focuses on the element, for example, clicks on <input>

Keyboard events

- keydown – when the visitor presses the key
- keyup – when the visitor releases a key

Assigning Event Handlers

An event can be assigned a handler, that is, a function that will work as soon as an event has occurred.

It is thanks to handlers that JavaScript code can react to visitor actions.

There are several ways to assign an event handler. Now we will consider them, starting from the simplest.

Using the HTML attribute

The handler can be assigned directly in the markup, in an attribute called on <event>.

For example, to attach a click event to the input button, you can assign an onclick handler, like so:

```
<input value="Нажми меня" onclick="alert('Click!')" type="button">
```

When you click the button, the code specified in the onclick attribute will be executed.

Using the DOM Object Property

You can assign a handler using the DOM element's on <event> property.

Example of setting the click handler:

```
<input id="elem" type="button" value="Click me" />

<script>

  elem.onclick = function() {

    alert( 'Thank you' );

  };

</script>
```

If the handler is specified via an attribute, the browser reads HTML markup, creates a new function from the attribute content, and writes it to the onclick property.

The handler can also assign an existing function:

```
function sayThanks(){

  alert( 'Thak you!' ); }

elem.onclick = sayThanks;
```

Accessing an item through this

Inside the event handler, this refers to the current element, that is, to the one on which it worked.

You can use this to get properties or change an element.

In the code below, the button displays its content using this.innerHTML:

```
<button onclick = "alert (this.innerHTML)"> Click me </ button>
```

Common mistakes

If you are just starting to work with events – pay attention to the following features.

The function must be assigned as **sayThanks**, not **sayThanks()**.

```
button.onclick = sayThanks;
```

If you add brackets, then sayThanks () will be the result of the function execution (and since there is no return in it, then onclick will get undefined). We also need a function.

But in HTML just the brackets are needed:

```
<input type="button" id="button" onclick="sayThanks()" />
```

This difference is easy to explain. When you create a handler with a browser from an attribute, it automatically creates a function from its contents. Therefore, the last example is actually the same as:

```
button.onclick = function() {

  sayThanks();

};
```

The DOM-property register has a value

При назначении через DOM нужно использовать свойство onclick, а не ONCLICK.

Lack of assignment through the property

The fundamental drawback of the above methods of assigning a handler is the impossibility of hanging several handlers on one event.

For example, one part of the code wants to make it highlighted while the button is clicked, and the other part – to issue a message. You need to hang two handlers in different places.

In this case, the new handler will overwrite the previous one. For example, the following code actually assigns one handler, the last one:

```
input.onclick = function () {alert (1); } // ...

input.onclick = function () {alert (2);} // replace the previous
handler
```

The developers of the standards have understood this for a long time and proposed an alternative way of assigning handlers using special methods that are free from this shortcoming.

addEventListener and removeEventListener

The addEventListener and removeEventListener methods are a modern way to assign or remove a handler, and at the same time allow you to use as many arbitrary handlers as you like.

The handler is assigned by calling addEventListener with three arguments:

```
element.addEventListener(event, handler[, phase]);
```

– event – The name of the event, for example click
– handle – A reference to the function to be set by the handler
– phase – Optional argument, the "phase" on which the handler should trigger. This argument is rarely needed.

Removal of the handler is done by calling removeEventListener, the function must pass the same arguments that addEventListener had.

```
element.removeEventListener(event, handler[, phase]);
```

To delete, you need to pass exactly the function-handler that was assigned. This is how removeEventListener does not work:

```
elem.addEventListener( "click" , function() {alert('Thank
you!')});

// ....

elem.removeEventListener( "click", function() {alert('Thank
you!')});
```

The removeEventListener is passed not the same function, but the other, with the same code, but it does not matter.

That's right:

```
function handler() {

  alert( 'Thank you!' );

}

input.addEventListener("click", handler);

// ....

input.removeEventListener("click", handler);
```

Pay attention – if the function is not saved anywhere, but simply passed to addEventListener, as in the previous code, then it will be impossible to get it back to remove the handler. There is no method that allows the event handlers assigned through addEventListener to be read.

Advantages of addEventListener

- Some events can only be assigned via addEventListener.
- The addEventListener method allows you to assign many handlers to one event.

Disadvantages of addEventListener

- The handler assigned via onclick is easier to remove or replace
- The onclick method is cross-browser.

Practical Work. DOM. Events

Change the HTML page index.html (see bellow) as follows.
Create a page index1.html (from index.html).
The amount in each column should be dynamically updated when a new switch value is selected.
The "Reset" button switches all the switches to "Not"
You can add JavaScript code only!. You can not change HTML tags!

```
<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>

<title>John Smith</title>

<style>

.class_01 {

  background-color: #FFFFE0;

}

.class_02 {

  background-color: #FF0000;

}

.class_03 {

  background-color: #CCCCCC;

}

</style>

</head>
```

```
<body>
<form name="dialForm">
<table width="100%" border="1" cellspacing="1" cellpadding="5">
  <tr>
    <th width="20" scope="col" bgcolor="#FFFFE0">OK</th>
    <th width="20" scope="col" bgcolor="#FF0000">Err</th>
    <th width="20" scope="col" bgcolor="#CCCCCC">Not</th>
    <th width="50" scope="col">Pic</th>
    <th scope="col">Text</th>
  </tr>
<tr>
    <td align = "center" valign = "top" class = "class_01"> <input type = "radio" name = "Res01" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input type = "radio" name = "Res01" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input type = "radio" name = "Res01" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img / im01.gif" /> </ td>
    <td valign = "top"> Multiple choice (closed question) with only one correct answer (the student puts a mark in one of the circles)
    </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input type = "radio" name = "Res02" value = "OK" /> </ td>
```

```
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res02" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res02" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im01.gif" /> </ td>

    <td valign = "top"> Multiple choice (closed question) with one
or more correct answers (the student puts a mark in one or more
boxes)

  </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res03" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res03" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res03" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im03.gif" /> </ td>

    <td valign = "top"> Alternative question (True / False) </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res04" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res04" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res04" value = "OK" checked /> </ td>
```

```
    <td align = "center" valign = "top"> <img src = "ttv_img /
im04.gif" /> </ td>

    <td valign = "top"> Numeric question </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res05" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res05" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res05" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im05.gif" /> </ td>

    <td valign = "top"> Calculated question </ td>

  </ tr>

<tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res06" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res06" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res06" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im06.gif" /> </ td>

    <td valign = "top"> Nested Answers </ td>

  </ tr>

  <tr>
```

```
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res07" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res07" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res07" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im07.gif" /> </ td>

    <td valign = "top"> The question of correspondence </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res08" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res08" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res08" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im08.gif" /> </ td>

    <td valign = "top"> Short answer (open question) </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res09" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res09" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res09" value = "OK" checked /> </ td>
```

```
    <td align = "center" valign = "top"> <img src = "ttv_img /
im09.gif" /> </ td>

    <td valign = "top"> Description </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res10" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res10" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res10" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im10.gif" /> </ td>

    <td valign = "top"> Essays </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res11" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res11" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res11" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im11.gif" /> </ td>

    <td valign = "top"> The goal in the image </ td>

  </ tr>

  <tr>
```

```
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res12" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res12" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res12" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im12.gif" /> </ td>

    <td valign = "top"> All or Nothing </ td>

  </ tr>

  <tr>

    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res13" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res13" value = "OK" /> </ td>

    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res13" value = "OK" checked /> </ td>

    <td align = "center" valign = "top"> <img src = "ttv_img /
im13.gif" /> </ td>

    <td valign = "top"> On the correspondence (displacement) </
td>

  </ tr>

  <tr>

    <td align="center" valign="top" bgcolor="#FFFFE0"><b><div
id="totOK">0</div></b></td>

    <td align="center" valign="top" bgcolor="#FF0000"><b><div
id="totERR">0</div></b></td>

    <td align="center" valign="top" bgcolor="#CCCCCC"><b><div
id="totNOT">13</div></b></td>
```

```
    <td align="center" valign="top"> </td>
    <td valign="top"><input name="Res" type="button"
value="Reset"/></td>
  </tr>
</table>
</form>
</body>
</html>
```

Self-examination questions

1. How to check whether the event was canceled?
2. What is an popup?
3. What are the two most important programming paradigms for the JavaScript developer?
4. What is functional programming?
5. What is the difference between class and prototype inheritance?
6. What are the pros and cons of functional and object-oriented programming?
7. When is class inheritance an appropriate choice?
8. When is it better to use prototypical inheritance?
9. What does "composition of objects better than inheritance of classes" mean?
10. What are the pros and cons of monolithic architecture and micro services?
11. What is asynchronous programming and why is it important in JS?

Навчальне видання

БОЧАРОВ Борис Петрович,
ЛЕВІКОВ Юрій Володимирович,
ВОЄВОДІНА Марія Юріївна

СКРИПТОВІ МОВИ ПРОГРАМУВАННЯ

Навчальний посібник

(Англ. мовою)

Відповідальний за випуск *М. В. Булаєнко*

За авторською редакцією

Технічний редактор *О. В. Михаленко*

Комп’ютерне верстання *Б. П. Бочаров*

Дизайн обкладинки *Т. А. Лазуренко*

Підп. до друку 24.12.2020. Формат 60 × 84/16.
Друк на різографі. Ум. друк. арк. 6,3.
Тираж 50 пр. Зам. №

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Маршала Бажанова, 17, Харків, 61002.
Електронна адреса: rectorat@kname.edu.ua
Свідоцтво суб’єкта видавничої справи:
ДК № 5328 від 11.04.2017.