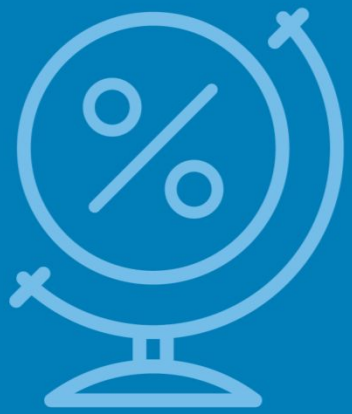
SoftServe

Empowering your Business through Software Development



Understanding JavaScript and Coding Essentials

Vyacheslav Koldovskyy

Last update: 27/08/2015

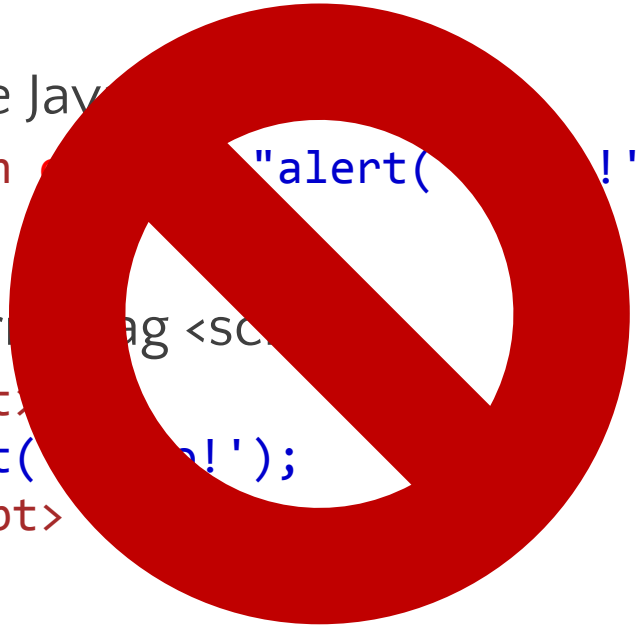
Agenda

- Basic Information
- How to include JS Code into HTML
- Comments
- Variables
- Data Types
- Type Casting
- Functions in JS
- Input and Output
- JS Code Processing
- Declaration and Expression

Basic information

- **JavaScript** – dynamic computer programming language.
- It is most commonly used as part of **web browsers**, whose implementations allow **client-side** to interact with the user, control the browser and asynchronously communicate with server-side.
- JavaScript syntax was influenced by **C**.
- **JS** supported object-oriented, imperative and functional programming styles.

How to add JavaScript to HTML?



1. Inline Jav

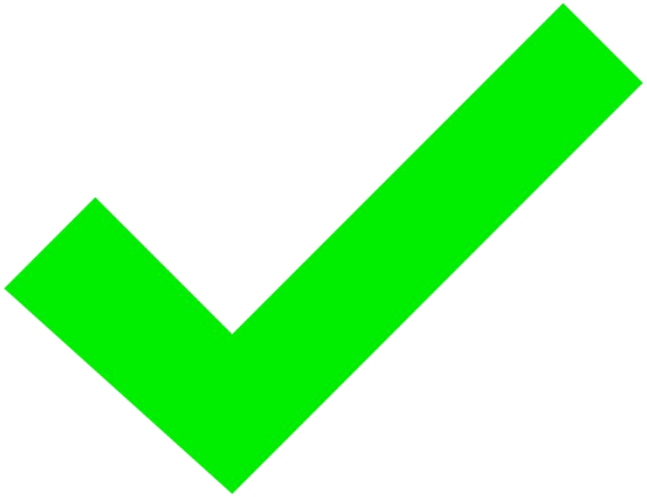
```
<button                "alert(          !');"></button>
```

2. Inter ag <sc

```
<script>
  alert(       !');
</script>
```

3. External file:

```
<head>
  <script src="script.js"></script>
</head>
```



Comments

Comments – part of the program text which will be ignored by language interpreter [1]

- The **/*** characters, followed by any sequence of characters (including new lines), followed by the ***/** characters. [2]
- The **//** characters, followed by any sequence of characters, but only in current line. Therefore, it is commonly called a "single-line comment." [3]

Variables

Variable – symbolic name associated with a value and whose associated value may be changed. [1]

Declaration – process of variable's specifying. Usually declaration consist of defining: type, name and default value of variable. [2]

A process in which a variable is set to its first value is called **initialization**. [3]

Declaration and initialization

var – special keyword for declaration of variables In JavaScript [1]

```
var variable;       //declaration
variable = 10;      //initialization
```

[2]

Or quickly

```
var variable = 10;
```

[3]

Global and local

JavaScript has two types of variables:

- **global** – exist in memory and is available at all times of the program. In JS it's a variables of page. [1]
- **local** – exist in memory and is available only in block when variable is defined. In JS it's defined in function variables. [2]

Data types

JavaScript has 6 base data types:

- **Number** – scalar type for integer and real digits
- **Boolean** – scalar type for logical values
- **String** – special type for work with text information
- **Undefined** – special type for uninitialized variables
- **Null** – special type for "cleaning" of variables
- **Object** – complex type for service and user needs

Number, Boolean and String

var n = 10; or **var n** = **Number**(10);

//number values for example: **-1, 10, 3.14, Nan, Infinity** [1]

var b = true; or **var b** = **Boolean**(true);

//bollean values: **true** and **false** [2]

var s = "text"; or **var s** = **String**("text");

//string values for example: **""**, **"text"**, **'text'** [3]

Null and Undefined

```
var n = null;
//null variables can have only null value
```

[1]

```
var u;
// created and uninitialized
```

And **Object** type... but it will be reviewed in future :)

Type casting

There are two types of casting:

```
var a, b, c;
a = 10;
b = true;
c = a + b;
```

[1] Explicit

Implicit [2]

```
var a, b, c;
a = 10;
b = true;
c = a + Number(b);
```

But both ways given **c =11** as a result! [3]

Type casting

Rules of typing casting:

[1] - All scalar types try to convert itself to largest scalar type: *Boolean* to *Number*, *Number* to *String*.

[2] - If Boolean converted to *String* it at first converted to *Number* and after them *Number* to *String*.

[3] - In mathematical operations (excluding **+**) *String* should be converted to *Number*.

[4] - *Null* and *Undefined* converted to *String* as “null” and “undefined”, and to Number as a 0 and NaN

Functions

In mathematics:

Function is a relation between a set of inputs and a set of permissible outputs. [1]

$y = f(x)$ [2]

In classical programming

Function is a named part of a code that performs a distinct service. [3]

Example

```
var i, base, power, result;          [1]

base = 2; power = 2; result = 1;     [2]

for(i = 0; i < power; i++) {
    result *= base;                  [3]
}
console.log(result);    [4]

base = 3; power = 4; result = 1;

for(i = 0; i < power; i++) {
    result *= base;                  [5]
}
console.log(result);
```

Function Declaration

```
function name (a, b) {
    return a + b;
}
```

[1]

* you can return one value only [2]
* **return** always interrupts the execution.
* place your **return** at the end of a function [3]

[3]

Function call

Call – operation for execution of function. [1]

( ) – operator for this action. [2]

Usually function can be **called** by name. [3]

Example

```
var out;

out = pow(2, 2);
console.log(out);

out = pow(3, 4);
console.log(out);

function pow (base, power) {
   var result = 1;
   for(var i = 0; i < power; i++) {
       result *= base;
   }
   return result;
}
```

Code processing

```
var a = 10;
test();
function test () {
    a = 30;
    var b = 40;
}
var b = 20;
console.log(a, b);
```

Code processing

```
var a = 10;
test();
1.  function test () {
        a = 30;
        var b = 40;
    }
var b = 20;
console.log(a, b);
```

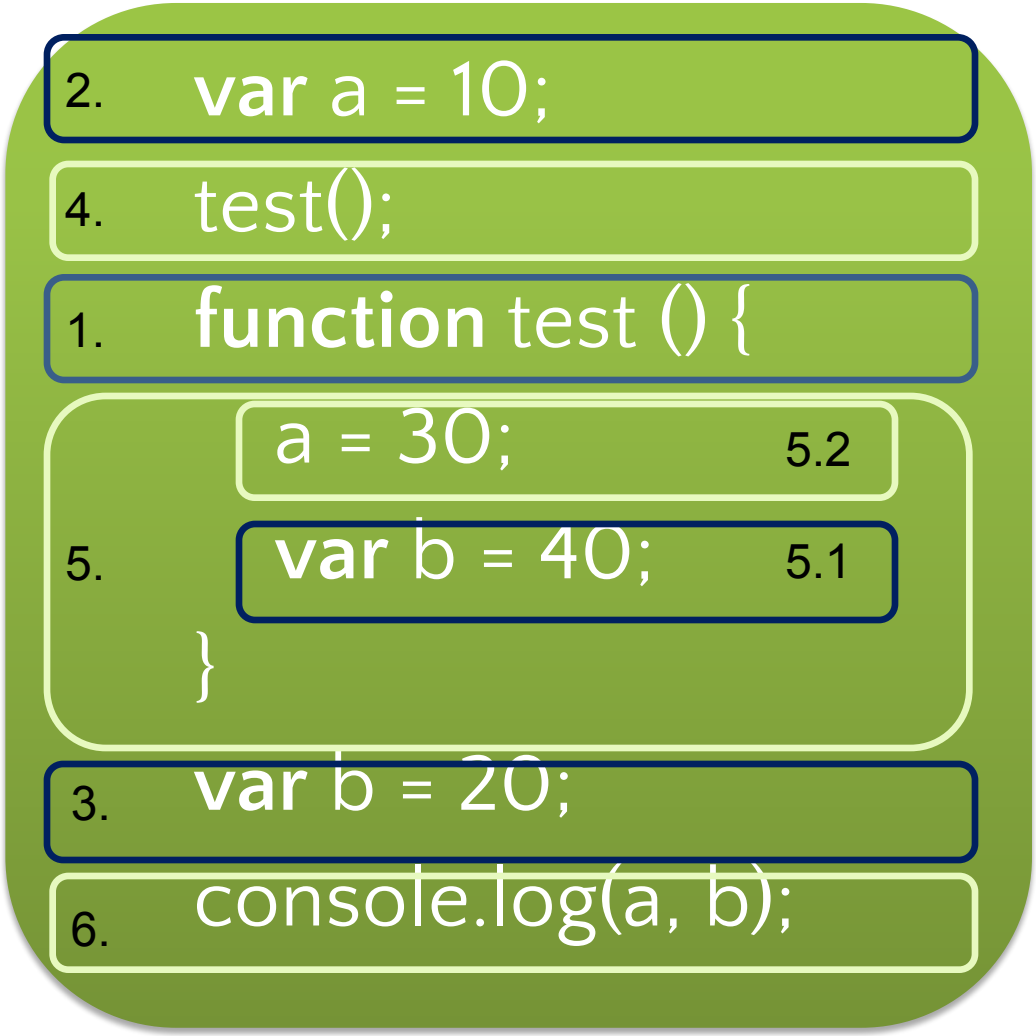
Code processing

```
2.  var a = 10;
    test();
1.  function test () {
        a = 30;
        var b = 40;
    }
3.  var b = 20;
    console.log(a, b);
```

Code processing

```
2.  var a = 10;
4.  test();
1.  function test () {
        a = 30;
5.      var b = 40;
    }
3.  var b = 20;
6.  console.log(a, b);
```

Code processing



Function Declaration and Expression

```
function name () {
    body;
}
```

[1]

```
var name = function () {
    body;
};
```

[2]

Additional Facts About Functions

Functions in JavaScript are Objects. [1]

As a result, functions are accessible by reference. [2]

Functions can be used as a parameter in other function. [3]

References to functions can be saved in any other variable. [4]

Program flow

Operators in a program processed in linear order: from top to bottom and from left to right.

[1]

Such sequence is called **Program flow**.

There are several methods intended to change standard flow. You already know about *function*. Also JavaScript has *conditions*, *loops* and *switch statement*.

[2]

Conditions: if-else

Very often we have to choose Most of algorithms have situation when next step related of some conditions depended on previous steps. It's a reason to use **if-else** statement. [1]

```
if (condition) {
    true branch;
} else {
    false branch;
}
```
[2]

```
if (condition) {
    true branch;
}
```
[3]

Conditions: if–else

Example:

Function get a parameter with a information about discount. And if discount is "silver" or "gold", function modifies global variable price.

In this example a shortened form of operator was used.

```
function discount (type) {
    if (type === “silver”) {
        price *= 0.9;
    }
    if (type === “gold”) {
        price *= 0.85;
    }
    return price;
}
```

Conditions: ?:

Sometimes *if-else* too bulky. If we need to initialize a variable modifying it by simple conditions; or we need to return a value from function and this value is dependent on something, we can use ternary

Ternary operator like **?:**.

```
result = (condition)? true action: false action;
```

[1]

Let's rewrite the last example using ternary operator.

Conditions: ?:

```
function discount (type) {
    if (type === “silver”) {
        price *= 0.9;
    }
    if (type === “gold”) {
        price *= 0.85;
    }
    return price;
}
```

We get a more compact but a less readable code. So be careful!

```
function discount (type) {
    price *= (type === “silver”)? 0.9: 1;
    price *= (type === “gold”)? 0.85: 1;
    return price;
}
```

Loops: for

Loops are used when algorithm requires repeating of statements. [1]

First of them: **for** – loop with counter

```
for (start position; repeat condition; step) {
    body of loop;    // will be repeated
}
```
[2]

One processing of loop's body is called **iteration**. [3]

Loops: while and do-while

Two others types of loops: **while** and **do-while**

```
while (condition) {
    body of loop;
}
```

[1]

```
do {
    body of loop;
} while (condition);
```

The main difference between these loops is the moment of condition calculation. *While* calculates condition, and if the result is true, *while* does iteration. *Do-while* initially does iteration and after that calculates a condition. [2]

Loops: examples

Example 1:

Text with number of current iteration will be print 5 times

```
for (var i = 0; i < 5; i++) {
    console.log("Iteration # %d", i + 1);
}
```
[1]

Example 2:

This loop will be repeated until accumulation reaches 100 or gets grater value.

```
while (accumulation < 100) {
    accumulation += doSomething();
}
```
[2]

Loops: break and continue

There are two keywords for loops control :

- **break** – aborts loop and moves control to next statement after the loop;
- **continue** – aborts current iteration and immediately starts next iteration.

Try not to use this keywords. A good loop have one entering point, one condition and one exit.

Switch

Switch statement allows to select one of many blocks of code to be executed. If all options don't fit, default statements will be processed

```
switch (statement) {
    case value1: some body;
                 break;
    case value2: some body;
                 break;
    . . .
    default: some body;
}
```

Switch

Example:

This switch looks for the word equivalent for a mark in the 5-point system

Default statement is not used.

```
switch (mark) {
    case 5: result = “excellent”;
            break;
    case 4: result = “good”;
            break;
    case 3: result = “satisfactorily”;
            break;
    case 2: result = “bad”;
            break;
}
```

Practice Task

Contacts

Europe Headquarters

52 V. Velykoho Str.
Lviv 79053, Ukraine

Tel: +380-32-240-9090
Fax: +380-32-240-9080

E-mail: info@softserveinc.com
Website: **www.softserveinc.com**

US Headquarters

12800 University Drive, Suite 250
Fort Myers, FL 33907, USA

Tel: 239-690-3111
Fax: 239-690-3116

Thank You!