



# 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 about JavaScript

# 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**.

# Basic information

**JS** take many names and naming conventions from **Java**, but the two languages are otherwise unrelated and have very different semantics. [1]

JavaScript is a **prototype-based** scripting language with dynamic typing. [2]

**JS** supported object-oriented, imperative and functional programming styles. [3]

# How to include JS Code into HTML

# Including of JavaScript

Exist three ways to include script into HTML page:

- Inline in HTML [1]
- Inside tag `<script>` [2]
- In separate file [3]

# Inline including

```
<button onclick = "f();" >.... </button>
```

Unfortunately, this is the worst solution. Holistic code will be broken into smaller parts. Such parts are difficult to test and maintain. :(



# Inside tag <script>

```
<script>  
  f() ;  
</script>
```

Sometimes it makes sense.

But in the general case, page size will be increased  
and, as a result, its performance reduced.

# In separate file

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

This is the best way.

Code is holistic. It's easy to test and maintain.

In addition, separated files are cached by the browser.

This improves performance. :)

# Comments

# 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

# 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 [1]  
In JavaScript

```
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

# Data types

JavaScript have 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);  
//boolean 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
```

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

[1]

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

# Type Casting

# Type casting

Exist 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

## Base 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 JS



# Basic Information

## 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++) {      [3]
    result *= base;
}

console.log(result);              [4]

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

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

console.log(result);
```

# Declaration of function

**function** is a special keyword for creation of function in JavaScript. [1]

```
function name () {  
    body; [2]  
}
```

# Example



```
var i, base, power, result;  
  
base = 2; power = 2; result = 1;  
  
for(i = 0; i < power; i++) {  
    result *= base;  
}  
console.log(result);
```



```
base = 3; power = 4; result = 1;  
  
for(i = 0; i < power; i++) {  
    result *= base;  
}  
console.log(result);
```

# Example

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

# Function call

**Call** – operation for execution of function. [1]

**()** – operator for this action. [2]

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

# Example

```
var base, power, result;

base = 2; power = 2;
pow();
console.log(result);

base = 3; power = 4;
pow();
console.log(result);

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

# Input and Output



# Input and Output

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

[1]

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

[3]

# Example

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

# Example

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

# 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;  
}
```

# JS Code Processing

# 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

2. `var a = 10;`

4. `test();`

1. `function test () {`

`a = 30;` 5.2

5. `var b = 40;` 5.1  
`}`

3. `var b = 20;`

6. `console.log(a, b);`

# Declaration and Expression

# 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]

# 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](mailto:info@softserveinc.com)

Website: **[www.softserveinc.com](http://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!