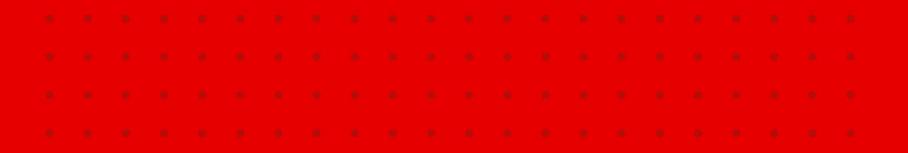


JavaScript fundamentals

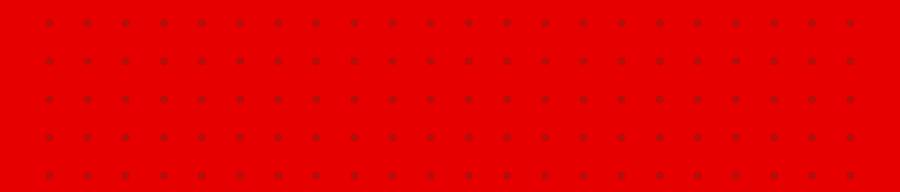


Pregatit de: Daniel Vișoiu Data: 04.05.2015

Agenda

- 1. Introduction
- 2. JavaScript basics
- 3. Data structures and types
- 4. Functions
- 5. Built-in types
- 6. Control flow and error handling
- 7. DOM interactions

Introduction



What is JavaScript?

- A cross-platform, object-oriented scripting language
- Small and lightweight
- The language for web pages
- Is **NOT** Java (but they do have some similarities)

History

- created in 1995 by Brendan Eich, an engineer at Netscape, as a way to add programs to web pages
- introduced in 1996 with the second version of Netscape Navigator browser
- Netscape submitted the language to Ecma International (European Computer Manufacturers Association), which resulted in the ECMAScript standard in 1997
- is one of the major implementations of ECMAScript (other implementations are **ActionScript** (Adobe), **JScript** (Microsoft))
- current stable release: 1.8.5 (March 2011)

теαтпет

Why use JavaScript?

It adds behaviour to the web page making it capable of responding to actions without needing to load a new web page:

- form validation
- loading new images, objects or scripts
- improving user experience

Basics

Basics

- JavaScript borrows most of its syntax from Java, but is also influenced by Awk, Perl and Python.
- Is **case-sensitive** and uses the Unicode character set
- Spaces, tabs and newline characters are called whitespace
- Instructions are called **statements** and are separated by a semicolon (;)
- Has rules for automatic insertion of semicolons (ASI) to end statements;
 but it is recommended to always add semicolons to end your statements (it will avoid side effects)

How to use JavaScript

External file – using the <script></script> tag

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

теαтпет

3.

The web console

- The Web Console shows you information about the currently loaded Web page
- includes a command line that you can use to execute JavaScript expressions in the current page.
- Usually opens with F12

Hello World

```
function hello(user) {
    return "Hello " + user;
}
hello("world"); // "Hello world"
```

Lazy version: console.log("Hello world");



Good to know before we start ·

• *alert(message)* – function that shows the given message in a small popup window (with an OK button)

```
alert('Hello world');
```

• *console.log(message)* – function that shows the given message in the **JavaScript console window** (that opens with F12 in most browsers)

```
console.log("Hello world");
```

- All the proposed exercises will be resolved using external .js files ©
- There will be no JavaScript code written in the HTML ©



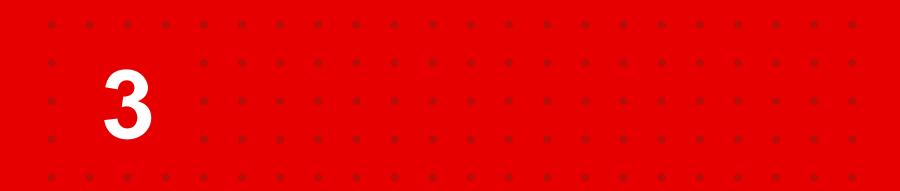
Comments

// single line comment

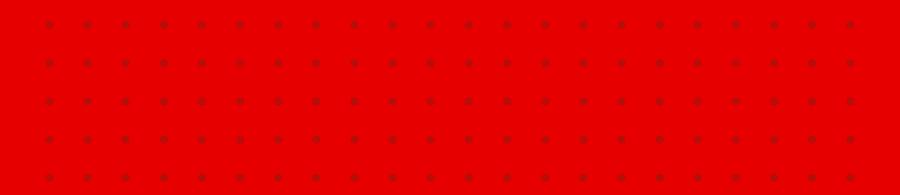
```
/*
multiline
comment
*/
```

alert("Hello World"); //comments can be appended to the end of lines

теαтпет



Data structures and types



Variables •

The names of variables, called **identifiers**, conform to certain rules:

- must start with a letter, underscore (_), or dollar sign (\$)
- subsequent characters can also be digits (0-9)
- case-sensitive

Some examples of legal names are Number_hits, temp99, and _name

Declarations

There are three kinds of declarations in JavaScript:

- 1. var
 - Declares a variable, optionally initializing it to a value.
- **2. let** (not fully supported)
 - Declares a block scope local variable, optionally initializing it to a value.
- **3. const** (not fully supported)
 - Declares a read-only named constant.

Data types

The latest **ECMAScript** standard defines seven data types:

- Six data types that are primitives:
 - **Boolean**: true and false
 - **null**: a special keyword denoting a null value. Because JavaScript is casesensitive, null is not the same as Null, NULL, or any other variant
 - **undefined**: a top-level property whose value is undefined.
 - **Number**: 42 or 3.14159
 - String: "Howdy"
 - **Symbol** (new in ECMAScript 6)
- Object



Data types

- The **primitives** enable you to perform useful functions with your applications
- Objects and functions are the other fundamental elements in the language. You can think of objects as named containers for values, and functions as procedures that your application can perform.

Data type conversion ·

- JavaScript is a **dynamically typed** language:
 - you don't have to specify the data type of a variable when you declare it
 - data types are converted automatically as needed during script execution.

var answer = 42; //defining a number variable

answer = "Thanks for all the fish..."; //reassingning the variable with a string value

Data type conversion ·

• In expressions involving numeric and string values with the + operator, JavaScript converts numeric values to strings.

```
x = "The answer is " + 42 // "The answer is 42" y = 42 + " is the answer" // "42 is the answer"
```

• In statements involving other operators, JavaScript does not convert numeric values to strings.

Converting strings to numbers

- In the case that a value representing a number is in memory as a string, there are methods for conversion.
- parseInt(string, radix)

radix = An integer between 2 and 36 that represents the base in mathematical numeral systems

parseFloat(string)

- **Scope** is the set of variables you have access to.
- There are two kinds of scopes

Local scope

- Variables declared within a JavaScript function, become LOCAL to the function.
- Local variables have local scope: They can only be accessed within the function.
- Local variables are created when a function starts, and deleted when the function is completed and they are no longer references.



Global scope

- A variable declared outside a function, becomes GLOBAL.
- A global variable has global scope: All scripts and functions on a web page can access it.

Automatically Global

If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable.

```
// code here can not use carName
function myFunction() {
   var carName = "Mercedes";

   // code here can use carName
}
```

Teamnet

```
var carName = "Mercedes";

// code here can use carName

function myFunction() {

    // code here can use carName
}
```

```
// code here can use carName
function myFunction() {
   carName = "Mercedes";

   // code here can use carName
}
```

Teamnet

Variable hoisting

- Another unusual thing about variables in JavaScript is that you can refer to a variable declared later, without getting an exception.
- This concept is known as **hoisting**; variables in JavaScript are in a sense "hoisted" or lifted to the top of the function or statement.
- However, variables that aren't initialized yet will return a value of undefined.

Variable hoisting

```
console.log(declaredLater);
// Outputs: undefined

var declaredLater = "Now it's defined!";

console.log(declaredLater);
// Outputs: "Now it's defined!"
```

Variable hoisting

```
console.log(getValue());
// Outputs: Hello world!

function getValue() {
    return "Hello world!";
}

console.log(getValue());
// Outputs: Hello world!
```

Literals ·

• You use **literals** to represent values in JavaScript. These are fixed values, not variables, that you literally provide in your script.

Literal integers:

- decimal (base 10) sequence of digits without a leading 0: 117 and -345
- octal (base 8) Leading 0 (zero) on an integer literal indicates it is in octal:
 015, 0001 and -077
- hexadecimal (base 16) Leading 0x (or 0X) indicates hexadecimal:
 0x1123, 0x00111 and -0xF1A7

String literals ·

- A string literal is zero or more characters enclosed in double (") or single (') quotation marks.
- A string must be delimited by quotation marks of the same type; that is, either both single quotation marks or both double quotation marks. The following are examples of string literals:

```
"foo"
'bar'
"1234"
"one line \n another line"
"John's cat"
```

теαтпет

Object literals

• An **object** literal is a list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({})

```
var sales = "Toyota";
var car = { myCar: "Saturn", cost: 15000, special: sales };
console.log(car.myCar); // Saturn
console.log(car.cost); // 15000
console.log(car.special); // Toyota
```

театтет

Equality

- Objects are only equal to themself
- Primitives are equal if the values match ("cat" === "cat")

.

- Two sets of equality operators (== and ===)
 - == performs type coercion if you give it different types
 - "dog" == "dog"; // true
 - 1 == true; // true
 - === avoids type coercion
 - 1 === true; // false
 - true === true; // true

Truthy and Falsy values

- The following values will evaluate to **false** (are falsy):
 - false
 - undefined
 - null
 - **(**)
 - NaN
 - the empty string ("")
- All other values, including all objects evaluate to true (are truthy)
- To test the Truthy/Falsy value of an *val* variable simply use double negation: console.log(!!val);

Exercise 1

- Create an object literal capable of storing the following information regarding a **hotel**:
 - *id* (unique identifier, integer)
 - *name* (string)
 - *description* (string)
 - *country* (string)
 - *city* (string)
 - *addedDate* (date)
 - *startPrice* (float)
- Output some of the properties to the browser console