Модуль 1

The minimum maintainable page

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8"/>

<title>The Smallest Page</title>

</head>

<body>

</body>

</html>

The code example above uses the DOCTYPE declaration for HTML5.

**<!DOCTYPE html>**

Transitional DOCTYPEs, which allow the use of deprecated, presentation-related elements from

previous versions of HTML.

**<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"**

**"http://www.w3.org/TR/html4/loose.dtd>**

• Frameset DOCTYPEs, which allow the use of frames in addition to the elements allowed by the

transitional DOCTYPE.

**<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"**

**"http://www.w3.org/TR/html4/frameset.dtd">**

• Strict DOCTYPEs, which do not permit the use of frames or deprecated elements from previous

versions of HTML.

**<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"**

**"http://www.w3.org/TR/html4/strict.dtd">**

**<body>**

**<h1 class="blue">An introduction to elements, tags and contents</h1>**

**<p>**

**<strong>Elements</strong> consist of <strong>content</strong> bookended by a**

**<em>start</em> tag and an <em>end</em> tag.**

**</p>**

**<hr />**

**<p>**

**Certain elements, such as the horizontal rule element, do not need content however**

**and consist of a single, self-closing element. These are known as empty elements.**

**</p>**

**</body>**

strong and em elements represent “strong importance" and “emphasis" for their contents, which browsers tend to render respectively as text in bold and text in italics.

Attributes provide additional information, presentational or semantic, about the contents of an element. They appear within the start tag of an element and have a name and a value. The name should be in lowercase characters

**<body>**

**<h1>An Introduction to HTML</h1>**

**<p>In this module, we look at the history of HTML and CSS.</p>**

**<h2>In the Beginning</h2>**

**<p>**

**WorldWideWeb was a piece of software written by Sir Tim Berners-Lee at CERN as a**

**replacement for**

**Gopher. It and HTML v1 were made open source software in 1993. The World Wide Web as**

**we know it**

**started with this piece of software.**

**</p>**

**<h3>Browser Wars</h3>**

**<p>The openness of WorldWideWeb meant many different web browsers were created early**

**on, including Netscape Navigator and NCSA Mosaic, which later became Microsoft Edge.</p>**

**</body>**

When writing HTML markup, remember that any sequence of whitespace characters–spaces, tabs, and

carriage returns–inside text are treated as a single space. The only exception to this is when that sequence

is inside a <pre> element, in which case the browser displays all the spaces.

**Adding stress to text**

<p>

To <strong>emphasize</strong> is to give extra weight to (a communication); <em>"Her

gesture emphasized her words"</em>.

</p>

**Unordered, ordered, and definition lists**

<body>

<p>Here’s a small list of HTML editors</p>

<ul>

<li>Notepad</li>

<li>Textmate</li>

<li>Visual Studio</li>

</ul>

<p>Here’s how to write a web page</p>

<ol>

<li>Create a new text file</li>

<li>Add some HTML</li>

<li>Save the file to a website</li>

</ol>

<p>Here’s a small list of people in the Internet Hall of Fame and what they did</p>

<dl>

<dt>Sir Tim Berners Lee</dt>

<dd>Invented HTML and wrote WorldWideWeb</dd>

<dt>Linus Torvalds</dt>

<dd>Originator of Linux</dd>

<dt>Charles Herzfeld</dt>

<dd>Authorized the creation of ARPANET, the predecessor of the Internet</dd>

</dl>

</body>

Writing nested lists

<body>

<ol>

<li>Lesson One: Introduction to HTML

<ol>

<li>The structure of an HTML page</li>

<li>Tags, Elements, Attributes and Content</li>

<li>Text and Images</li>

<li>Forms</li>

</ol>

</li>

<li>Lesson Two: Introduction to CSS</li>

<li>Lesson Three: Using Visual Studio 2017</li>

</ol>

</body>

**Adding an image to a web page**

<body>

<p>

<img src="logo.jpg" alt="My Web site logo" height="100" width="100" />

</p>

<h1>Welcome to my site!</h1>

</body>

<https://www.w3.org/community/respimg/>

• src

• alt

• title

• longdesc

• height and width

• The src attribute specifies a URL that identifies the location of the image to be displayed

• The alt attribute identifies a text alternative for display in place of the image if the browser is still downloading it or cannot display it for some reason; for example, if the image file is missing. It typically describes the content of the image.

• The title attribute identifies some text to be used in a tool tip when a user’s cursor hovers over the image • The longdesc attribute identifies another web page that describes the image in more detail

• The height and width attributes set the dimensions in pixels of the box on the web page that will contain the image; if the dimensions are different from those of the image, browsers will resize the image on the fly to fit the box.

Only the src attribute is mandatory.

Anchor tags have the following **non-global attributes**:

**• href**

**• target**

**• rel**

**• hreflang**

**• type**

• The href attribute identifies the web page or resource to link to

• The target attribute identifies where the browser will display the linked page; valid values are \_blank,

\_parent, \_self, and \_top

• The rel attribute identifies what kind of link is being created

• The hreflang attribute identifies the language of the linked resource

• The type attribute identifies the MIME type of the linked resource

**Adding hypertext links to your web page**

<body>

<ul>

<li><a href="default.html" alt="Home Page">Home</a></li>

<li><a href="about.html" alt="About this Web site">About</a></li>

<li><a href="essays.html" alt="A list of my essays">Essays</a></li>

</ul>

</body>

The href attribute is the most important part of linking one online resource to another. You can use several different value types:

• A URL in the same folder (for example: about.html)

• A URL relative to the current folder (for example: ../about.html)

• A URL absolute to the server root (for example: /pages/about.html)

• A URL on a different server (for example: <http://www.microsoft.com/default.html>)

• A fragment identifier or id name preceded by a hash (for example: #section2)

• A combination of URL and fragment identifier (for example: about.html#section2)

Use the HTML **<form>** element to identify an area of your web page that will act as an input

form. This element has the following attributes:

• action

o method

o GET

o POST

• accept-charset

• enctype

• target \_blank, \_parent, \_self, and \_top.

• The action attribute, which identifies the URL of the page to which the form data submitted

by the user will be sent for processing.

o The method attribute, which defines how the data is sent to the server. Valid values are:

o GET for HTTP GET. This is the default value, but is not secure

o POST for HTTP POST. This is the preferred value

• The accept-charset attribute, which identifies the character encoding of the data submitted in the

form by the user.

• The enctype attribute, which identifies the MIME-type used when encoding the form data for

submission when the method is POST.

• The target attribute, which identifies where the browser will display the action page; valid values are

\_blank, \_parent, \_self, and \_top.

**text (default)**  A single-line text box

**password** A single-line text box where the text entered into the box is replaced by asterisks.

**Hidden** A field not visible to the user

**checkbox** A checkbox. Provides a yes/no or true/false choice. Use the selected attribute to

indicate if it is checked by default.

**Radio**  A radio button control. Use the name attribute to group several radio button

controls together. The form will allow either none or one of the grouped radio

buttons to be selected.

**Reset** A reset button. Clicking this resets all the fields to their initial values.

**submit** A submit button. Clicking this will submit the current form values to the action page for processing.

**Image** An image for use as a submit button. Use the src attribute to identify the image to

be used.

**Button** A button. This has no default behavior and may be used to run a script when

clicked, for example.

**File** A file control. Provides a way to submit a file to the server when the submit button

is clicked.

**<textarea>**

**<select>**

**<button>**

<textarea>, which generates a free-form, multiline, plain text edit box; use the rows and cols

• <select>, which defines a list box or drop-down list. Use the multiple attribute to indicate if the user can select more than one item from the list and <option> elements nested within <select> to identify the items. Use the <option>’s selected attribute to indicate that it is selected by default and

its value attribute to indicate a value other than its text content to be sent to the server when the form is submitted.

• <button>, which defines a button. Use the type attribute to indicate whether it is a submit, reset, or button (does nothing) button. The default is submit.

Form Layout Elements

<p>

<div>

• **<fieldset>**

<legend> element, which must be the first child of the <fieldset> element.

• **<label>**

**Using a form to obtain the details of a user**

<form method="post" action="/registration/new" id="registration-form">

<label for="first-name">First name:</label><br />

<input type="text" id="first-name" name="FirstName"/><br />

<label for="last-name">Last name:</label><br />

<input type="text" id="last-name" name="LastName"/><br />

<label for="email-address">Email address:</label><br />

<input type="email" id="email-address" name="EmailAddress"/><br />

<label for="password">Choose a password:</label><br />

<input type="password" id="password" name="Password"/><br />

<label for="confirm-password">Confirm your password:</label><br />

<input type="password" id="confirm-password" name="ConfirmPassword"/><br />

<label for="website">Website/blog:</label><br />

<input type="url" id="website" name="WebsiteUrl" /><br />

<button type="submit">Register</button>

</form>

**Attaching Scripts to an HTML Page**

• Write the JavaScript on the page as the content part of a <script> element.

**<script type="text/javascript">**

**alert('I am a line of JavaScript');**

**</script>**

• Save the JavaScript in a separate file on your web site and then reference it by using the src attribute

of the <script> element.

**<script type="text/javascript" src="alertme.js"></script>**

• Reference a third-party JavaScript file on a different web site.

**<script type="text/javascript"**

**src="http://ajax.contoso.com/ajax/jQuery/jquery-1.7.2.js">**

**</script>**

The <script> element has three attributes:

• **type**

• **src**

• **charset**

**The <noscript> element**

<body>

<noscript>This page uses JavaScript. Please enable it in your browser</noscript>

…

Rest of page

…

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

</body>

**Some simple CSS rules**

/\* Targets level 1 headings and renders them as large pink text using the Segoe UI font \*/

h1 {

font-size: 42px;

color: pink;

font-family: 'Segoe UI';

}

/\* Targets emphasized text, rendering it as italicized on a yellow background \*/

em {

background-color: yellow; /\* Yellow is a good highlight color \*/

font-style: italic;

}

**Introducing the element, class, and id selectors**

h2 {

font-size: 24px;

}

.red {

color: red;

}

#alert {

background-color: red;

color: white;

}

**Combining selectors**

h2.blue {

color: blue;

}

h2#toc {

font-weight: bold;

}

The selector, h2.blue returns the set of <h2> elements with the class "blue", and h2#toc returns the set of

<h2> elements with id "toc".

Note that these two sets may intersect, in which case the CSS properties and values for both rules will

apply—in this case to the set of <h2> elements with id "toc" and class "blue".

|  |  |
| --- | --- |
| h2.blue | Returns any <h2> elements of class "blue" |
| h2#blue | Returns any <h2> elements with id "blue" |
| section, h2 | Returns any <h2> and any <section> elements |
| section h2 | Returns any <h2> elements nested within a <section> element at any level. |
| section > h2 | Returns any <h2> elements nested immediately under a <section> element |

|  |  |
| --- | --- |
| section + h2 | Returns any <h2> elements immediately following and sharing the same parent element as a <section> element |
| section ~ h2 | Returns any <h2> elements following and sharing the same parent element as a <section> element |

Set of all elements within an <aside> element and makes them slightly fainter.

**aside \* { opacity : 0.6; }**

|  |  |
| --- | --- |
| input[type] | Returns any <input> elements that use the type attribute, whatever its value. |
| input[type="text"] | Returns any <input> elements where the type attribute value is exactly equal to the string "text". |
| input[foo~="red"] | Returns any <input> elements where the foo attribute (for instance, the class attribute) contains a space-separated list of values, one of which is exactly equal to "red". |
| input[type^="sub"] | Returns any <input> elements where the type attribute value begins exactly with the string "sub". |
| input[type$="mit"] | Returns any <input> elements where the type attribute value ends exactly with the string "mit". |
| input[type\*="ubmi"] | Returns any <input> elements where the type attribute value contains the substring "ubmi". |
| input[foo|="en"] | Returns any <input> elements where the foo attribute value is either exactly "en" or begins exactly with "en-", i.e. the value plus a hyphen. |

**The Attribute Selector**

You can combine attribute selectors by concatenating them.

For example, to return a set of all checkboxes that are checked by default, you would use the following selector:

**input[type="checkbox"][selected] {}**

body {

font-family: Candara;

}

vs

h1, h2, h3, h4, h5, h6 {

font-family: Candara;

}

p {

font-family: Candara;

}

…

Importance. You can ensure a certain property is always applied to a set of elements by appending

the rule with !important.

**h2 { font-weight : bold !important; }**

<https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Cascade_and_inheritance>

* Write rules specific to an element within its style attribute.

**<p style="font-family : Candara; fontsize: 12px; "> ... </p>**

* Write a set of rules specific to a page within its <head> element by using <style> tags.

**<style type="text/css">**

**p {**

**font-family : Candara; font-size: 12px;**

**}**

**</style>**

• Write all your rules in a separate style sheet file with the .css extension, and then reference it in the

markup of the page by using a <link> tag. The most common place to add a <link> tag is within the

<head> element.

**<link rel="stylesheet" type="text/css" href="mystyles.css" media="screen">**

The <link> element has four CSS-relevant attributes:

• The **href** attribute specifies a URL that identifies the location of the style sheet file.

• The **rel** attribute indicates the type of document the <link> element is referencing; set this to style

sheet when linking to style sheets.

• The **media** attribute indicates the type of device targeted by the style sheet; possible values include

**speech** for speech synthesizers, **print** for printers, **handheld** for mobile devices and all (the default),

indicating the style sheet is all purpose

• The **type** attribute indicates the MIME type of the document being referenced; the correct type for

style sheets is text/css which is also the default value for this attribute

The type and media attributes have the same function for the <style> element as their namesakes for the <link> element.

**Module 2 Creating and Styling HTML Pages**

HTML Design Principles

<https://www.w3.org/TR/html-design-principles/>

**Content Structure in HTML5**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>My Best Recipes</title>

</head>

<body>

<nav>

<a href="/">Home</a>

</nav>

<header>

<h1>My Best Recipes</h1>

<p>My favorite recipes</p>

</header>

<article>

<h1>Beans On Toast</h1>

<section>

<h1>Ingredients</h1>

<ul>

<li>Beans</li>

<li>Bread</li>

</ul>

</section>

<section>

<h1>Method</h1>

<ol>

<li>Toast bread</li>

<li>Heat beans</li>

<li>Put beans on the toast</li>

</ol>

</section>

</article>

<footer>

<small>Last updated on <time datetime="2012-08-12">August 12, 2012</time></small>

</footer>

</body>

</html>

<hgroup>

<h1>My Recipes</h1>

<h2>Great to eat, easy to make</h2>

</hgroup>

<time datetime="2012-08-08">Today</time>

<time datetime="2012-08-08T09:00:00-0500">9am today in New York</time>

<time>4h</time>

<time>2012</time>

<p>This text should be <mark>noted for future use</mark> rather than

<em>emphasized</em>.</p>

<p>Heat your beans for five minutes. <small>Or until they are hot enough for

you.</small></p>

<figure>

<img src="plateofbeans.jpg" alt="A Plate of beans on toast" />

<figcaption>A wonderful plate of beans in five minutes flat</figcaption>

</figure>

Раньше Ctrl+K+F - выравнивание кода

**Lesson 2 Styling an HTML5 Page**

font-family : Arial, Candara, Verdana, sans-serif;

font-family : Georgia, Corbel, "Times New Roman", serif;

font-family : Consolas, "Courier New", monospaced;

font-size

font-size : 16px;

font-size : 150%; /\* Font-size of the parent element \* 150% \*/

font-size : 1em; /\* 1em = base font-size of the page. Usually 16px \*/

font-weight : bold;

font-weight : normal;

font-weight : 800;

p { font : bold 16px/1.5 "Arial"; }

/\* The above is a shorthand for the following rules The default font-style is used.

\*/

p {

font-weight: bold;

font-size: 16px;

line-height : 1.5em;

font-family: Arial;

}

CSS also provides a shortcut property simply called font, which enables you to set some or all of these

four properties (plus line-height) in a single rule rather than having to write out all five rules for every

element. You must set the value for these properties in the following order (note that the font-family

and font-size properties are mandatory, but the other properties are optional):

1. font-style

2. font-weight

3. font-size/line-height

4. font-family

/\* The following color values are all equivalent. \*/

color : olive;

color : #808000;

color : rgb(128, 128, 0);

p {

opacity : 0.6;

filter:alpha(opacity=60); /\* IE8 and earlier \*/

}

letter-spacing : 2em;

letter-spacing : -3px;

line-height : 16px;

line-height : normal; /\* This is the default \*/

line-height : 1.2;

line-height : 120%;

text-align : left;

left, right, or justify

text-decoration : underline;

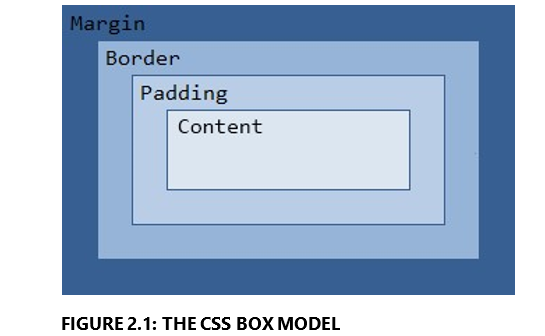
none (the default), underline, overline, line-through

text-transform : lowercase;

(the default), capitalize, uppercase, lowercase

Marging – поле

Padding – заполнение



border property to set its width, color, and style

**Using the box model properties**

h2.highlight {

height : 100px;

width : 500px;

padding : 10px;

border : 2px dotted blue;

margin : 25px 0 25px 0; /\* Could also be written 25px 0 \*/

}

padding-top: 10px;

padding-right : 10px;

padding-bottom : 10px;

padding-left : 10px;

Reader Aid: You can easily recall the order of the sides by thinking of the word TRouBLe: Top, Right, Bottom, and Left.

border-width: 2px;

border-style: dotted;

border-color: blue;

p.example {

padding: 10px;

border-bottom: solid 1px black;

border-left-style: dotted;

border-left-width: 2px;

}

* Visibility
* Display
* Position
* Float
* Overflow
* box-sizing (content-box (the default), border-box)

background-image:url('../images/pattern.jpg');

background-size: 40px 60px; /\* 40px wide, 60px high \*/

background-color : green;

background-color : #00FF00;

background-color : rgb(0, 255, 0);

background-position : left top; /\* Image locked into top left corner of element \*/

background-position : 100% 100%; /\* Image locked into bottom right corner of element \*/

background-position : 8px 8px; /\* Image starts 8px from left and 8px from top of element \*/

background-origin : border-box; /\* Possible values are content-box (the default), paddingbox, and border-box. \*/

background-repeat : repeat-x; /\* Repeat background image only horizontally \*/

background-repeat : no-repeat; /\* Don't repeat the image \*/

background-position : fixed;

1. background-color

2. background-position

3. background-size

4. background-repeat

5. background-origin

6. background-clip

7. background-attachment

8. background-image

For example:

article { background : transparent repeat-x url('fluffycat.jpg'); }

/\* The above is a shorthand for the following rules \*/

article { background-color : transparent; background-repeat : repeat-x; background-image : url('fluffycat.jpg'); }

Что такое rem?

Что такое div?

**Module 3 Introduction to JavaScript**

**Statements and comments**

<script type="text/javascript">

document.write("I'm learning JavaScript"); // display a message to the user

/\* You can use a multi-line comment

to add more information \*/

alert("I'm learning JavaScript too!");

</script>

There are two important rules for naming variables in JavaScript:

1. Variable names must begin with a letter or the underscore character.

2. Variable names are case sensitive.

**String:** Any set of characters (alphanumeric and punctuation) enclosed in single or double quotes.

To include special characters such as ",´, ;, \, and & in your string, escape them with a backslash.

Also use a backslash to split a string over two or more lines.

let simple = "Green Eggs and Ham";

let escaped = "\"Green Eggs \& Ham \"";

let verylong = "Cracked, fried, overripe ovoids and \

porcine strips cooked medium well and allowed to cool";

2. **Number**: Any integer or decimal number. Do not wrap a number between double quotes when you declare a numeric variable or it will be treated as a string.

let answer = 42;

let actuallyAString = "42"; // not treated as a number

3. **Boolean**: A Boolean value: true or false.

let canYouReadThis = true;

**Note**: Remember that if you declare a variable but do not give it a value, the variable is undefined.

You can also declare a variable and set it to null, like this:

let variableWithNullvalue = null;

Setting a variable to null indicates that a value does not exist, rather than that a variable has not been given a value. It is important to understand the difference.

let data = 99;

...

if (typeof data == "number") {

// data is numeric

}

1. **Arithmetic operators** indicate a mathematical function to be performed on values/variables:

+ (addition)

- (subtraction)

\* (multiplication)

/ (division)

% (modulus)

++ (increment)

-- (decrement)

1. **Assignment operators** assign values to JavaScript variables:

x= y

x += y (x = x + y)

x -= y (x = x - y)

x \*= y (x = x \* y)

x /= y (x = x / y)

x %= y (x= x % y)

1. **Comparison operators** determine if two values/variables are or are not equal.

The first set of comparison operators converts the two values/variables to the same type before comparison. == (is equal to)

!= (is not equal to)

> (is greater than)

< (is less than)

>= (is greater than or equal to)

<= (is less than or equal to)

The second set of two comparison operators does not convert the two values/variables to the same type before comparison.

=== (is equal in value and in type)

!== (is not equal in value or in type)

1. **Boolean operators** are used to perform Boolean operations.

x && y returns true if x and y are both true, false otherwise.

x || y returns true if either x or y or both are true, false otherwise.

!x returns true if x is false, false otherwise.

1. The **ternary conditional operator:**

?: assigns one of two values to a variable based on a condition.

For example, the

**expression x =(condition)?value1:value2;**

sets x to value1 if condition is true, value2 otherwise

1. **The string operator:**

+ concatenates two strings. For example, "Bo" + "om" returns "Boom".

There are a number of issues to be aware of with respect to JavaScript operators and how they convert

values\variables between types as the JavaScript interpreter executes expressions.

• If you add a number and a string, the result will be a string!

x=10 + 10; // x is set to the number 20;

y="10"+10; // y is set to the string "1010";

z="Ten"+10; // x is set to the string "Ten10";

0, "" (the empty string), undefined, and null all evaluate to false in Boolean operations. Always use

=== when comparing to any of these values.

let zero = 0;

let emptyString = "";

let falseVar = false;

zero == falseVar; // returns true;

zero === falseVar; // returns false;

emptyString == False; // returns true;

emptyString === False; // returns false;

Проверить что такое False.

function aName( argument1, argument2, …, argumentN ) {

statement1;

statement2;

…

statementN;

}

**Note:** Function arguments are optional. If you don't specify any arguments, you can still pass parameters into a function.

The arguments are available in an array called arguments. You can access the first argument by using the expression **arguments[0]**, the second argument by using the expression **arguments[1]**, and so on.

This mechanism gives you a way to define methods that can take a variable number of parameters. You can find out how many parameters were passed in by querying the value of **arguments.length**.

Creating and calling a function

function CalculateBill(numberOfNightsStay, nightlyRate, extras) {

return (numberOfNightsStay \* nightlyRate) + extras;

}

…

// elsewhere in the script

const TotalAmountOwed = CalculateBill(10, 100, 50);

**Note**: Not all functions have a name. You can even declare anonymous functions. You typically use anonymous functions when writing code to handle events or implement callbacks.

In these cases, the function is invoked by the browser (or whatever environment your code happens to be running in) rather than by your code, and it is referenced by a variable rather than its name.

**Conditional Statements**

if (TotalAmountOwed > AdvancePaid) {

GenerateNewInvoice(); // runs if condition is true

}

if (TotalAmountOwed > AdvancePaid) {

GenerateNewInvoice(); // runs if condition is true

} else {

WishGuestAPleasantJourney(); // runs if condition is false

}

let RoomRate;

switch (typeOfRoom) {

case "Suite":

RoomRate = 500;

break; // Use break to prevent code in next case statement being run.

case "King":

RoomRate = 400;

break;

default: // code to be executed if typeOfRoom does not match above cases.

RoomRate = 300;

}

**Looping Statements**

while (GuestIsStillCheckedIn())

{

numberOfNightsStay += 1;

}

do {

eatARoundOfToast();

} while (StillHungry())

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

plumpUpAPillow();

}

Break

Справочник JavaScript

[**https://html5css.ru/jsref/default.php**](https://html5css.ru/jsref/default.php)

[**https://itproger.com/spravka/javascript**](https://itproger.com/spravka/javascript)

[**https://codernet.ru/books/js/**](https://codernet.ru/books/js/)

**Using Object Types**

let eventWelcome = new String('Welcome to your conference');

let len = eventWelcome.length;

let today = new Date(1346454000); // Number of milliseconds since 01/01/1970

let today = new Date("September 1, 2012");

let today = new Date(2012, 8, 1); // Note January is 0, .., December is 11.

let emptyThreeItemArray = new Array(3);

let seasonsArray = new Array("Spring", "Summer", "Autumn", "Winter");

let thirdSeason = seasonsArray[3]; // Winteris 11.

let seasonsArray = ["Spring", "Summer", "Autumn", "Winter"];

let autumnLocation = seasonsArray.indexOf("Autumn");

let re = new RegExp("[dh]og");

if (re.test("dog")) {...}

The Math object gives you access to various mathematical constants (for example, Pi and E) and functions (sine, cosine, square root, and a pseudo-random number generator) as static properties and methods.

For example, Math.E, Math.cos(); let seed = Math.random(); • The Global object contains global functions and constants and is the parent object for the undefined, NaN, and Infinity constants. It cannot be instantiated.

++

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects>

https://jpoint.ru/?utm\_source=ya&utm\_medium=rsya&utm\_campaign=rsya\_jpoint20&term=%D1%8F%D0%B7%D1%8B%D0%BA%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%20java&utm\_content=4&yclid=258391170708958926

let myObject = {

"propertyName1" : "propertyValue1",

"propertyName2" : "propertyValue2",

… ,

"propertyNameN" : "propertyValueN"

};

let singleAttendee = { "name" : "Eric Gruber",

"currentTrack" : "1" };

let listOfAttendees = [

{ "name": "Eric Gruber", "currentTrack": "1" },

{ "name": "Martin Weber", "currentTrack": "2" }

];

JavaScript - JSON и методы работы с ним

<https://itchief.ru/lessons/javascript/javascript-json>

console.log(`${array[i].name} (${array[i].age}) ${array[i].email}`);

<https://learn.javascript.ru/string>

**Finding Elements in the DOM**

* forms
* images
* links
* anchors
* applets

<form name="contactForm">

<input type="text" name="nameBox" id="nameBoxId" />

</form>

document.forms[0] // forms is a zero-based array

document.forms["contactForm"]

document.forms.contactForm

document.contactForm

document.forms.contactForm.elements[0]

document.forms.contactForm.elements["nameBox"]

document.forms.contactForm.nameBox

document.contactForm.nameBox

document.getElementById( IdString )

document.getElementsByName( NameString )

let userNameBox = document.getElementById("nameBoxId");

let username = userNameBox.value;

**Querying the DOM for Elements**

document.querySelector(cssSelectorString)

document.querySelectorAll(cssSelectoreString)

let validInput = document.querySelector("input.valid"); /\* элемент input class valid\*/

let validInput = validInput.value;

let checkedInputs = document.querySelectorAll("input[checked]"); /\* all input elements that have the checked attribute.\*/

let checkedInputsCount = checkedInputs.length;

let mainDV = document.querySelector("div#main");

let boldSpans = mainDV.querySelectorAll("span.text-bold");

let boldSpansCount = boldSpans.length;

/\* The following example shows how to obtain reference for all the **span** elements where the class attribute has the value **text-bold**, and are **contained inside div** where the **ID attribute has the value main**\*/

The **querySelector** and **querySelectorAll** methods also exist in the Element object which is the base class from which all objects in Document are inherited. This enables us to only query against the contents of a specific Element object.

**Adding, Removing, and Manipulating Objects in the DOM**

DOM Core API defines several methods to create new objects:

**• document.createElement(tagname)**

**• document.createTextNode(string)**

**• document.createAttribute(name, value)**

**• document.createDocumentFragment**

**document.getElementbyId()** to retrieve the parent element to which

you wish to apply this object, and then call one of the following methods on this element:

• **appendChild(newNode)**, which adds the new node as the last child of the selected element.

• **insertBefore(newNode, existingNode)**, which adds the new node into the DOM before but as a

sibling to the given existingNode.

• **replaceChild(newNode, existingNode)**, which replaces the existing child node with the new node.

• **replaceData(offset, length, string)**, which replaces the text in a text node. The offset parameter

specifies which character to begin with, length specifies how many characters to replace, and string

specifies the text to insert.

* **childNodes**, which returns all the child nodes of a node.
* **firstChild**, which returns the first child of a node.
* **lastChild**, which returns the last child of a node.
* **nextSibling**, which returns the node immediately following the current one.
* **parentNode**, which returns the parent node of a node.
* **previousSibling**, which returns the node immediately prior to the current node.

**Modifying a list**

<!-- HTML Markup for VenueList -->

<ul id="VenueList">

<li>Room A</li>

<li>Room B</li>

</ul>

// JavaScript code to modify the items in VenueList

const list = document.getElementById("VenueList");

// Create a new venue

const newItem = document.createElement("li");

newItem.textContent = "Room C";

// Add the new venue to the end of VenueList

list.appendChild(newItem);

The DOM also defines methods for removing nodes from the document tree, including:

• **removeChild(node)**, which removes the target node.

document.removeChild(

document.getElementById("VenueList").firstChild

);

• **removeAttribute(attributeName)**, which removes the named attribute from the element node.

const list = document.getElementById("VenueList");

list.removeAttribute("id");

• **removeAttributeNode(node)**, which removes the given attribute node from the element.

const list = document.getElementById("VenueList");

list.removeAttribute(list.attributes[0]);

To clear a text node rather than removing it completely, just set it to an empty string.

**Handling Events in the DOM**

<img src="helpicon.gif" id="helpIcon" onmouseover="window.alert('Some help text');"

/>

**Note**: A callback is a reference to a function that runs as the result of another action completing. In the case of an event handler for an HTML element, the browser causes the callback to run when it triggers the corresponding event.

document.images.helpIcon.onmouseover = function() { window.alert('Some help text'); };

**addEventListener(eventName, listenerFunction, useCapture)**, which adds the listener function to the element for the given eventName.

You can pass the listenerFunction by name or as an anonymous function.

let helpIcon = document.getElementById("helpIcon");

// Add an event listener for the mouseover event

// by using a named function

Function ShowHelpText()

{

window.alert('Some help text');

}

helpIcon.addEventListener("mouseover", ShowHelpText, false);

// Alternatively, using an anonymous function

helpIcon.addEventListener("mouseover",

function() { window.alert('Some help text'); }, false);

• **removeEventListener(event, listenerFunction, useCapture)**, which removes the listener function

from the element for the given eventname.

helpIcon.removeEventListener("mouseover", ShowHelpText, false);

Что такое **useCapture**?

**useCapture**, allows developers to choose if they want to trigger the event handler in the capturing phase by setting it to true or in the bubbling phase by setting it to false. The default value of this argument is false.

**Binding an action to an event with the DOM**

<!DOCTYPE html>

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

<head>

<meta charset="utf-8" />

<title>Binding events with JavaScript</title>

</head>

<body>

<form>

<p>

<label>Write Your Name:

<input type="text" id="NameBox" /></label></p>

<input type="button" id="submit" value="Click to submit" />

</form>

<div id="thankYouArea"></div>

<script type="text/javascript">

function sayThankYou() {

let userName = document.getElementById("NameBox").value;

let thankYou = document.createElement("p");

thankYou.textContent = "Thank you " + userName;

document.getElementById("thankYouArea").appendChild(thankYou);

}

document.getElementById("submit").addEventListener("click", sayThankYou);

</script>

</body>

</html>

The DOM is constructed by the browser while parsing the HTML markup from top to down. Some elements may block the construction of the DOM; the script element is one of these elements.

When the browser hits a script element, the construction of the DOM is blocked until the script file is loaded and executed. Consider the following example:

<script type="text/javascript">

const main = document.getElementById("mainContent");

console.log(main) //will output null

</script>

<div id="mainContent" >Some content</div>

In the code example above, **document.getElementById** will return **null** because the element where the id attribute has the value mainContent does not exist in the DOM when the script is run.

The above situation can be avoided by **listening to the document’s DOMContentLoaded event.**

<script type="text/javascript">

function init(){

const main = document.getElementById("mainContent");

console.log(main) //will output the div element reference object.

}

document.addEventListener("DOMContentLoaded",init);

</script>

<div id="mainContent" >Some content</div>

Что такое MVC?

Что такое bootstrap?

Почитать про const

Как найти Design View

<https://itchief.ru/lessons/javascript/ajax-asynchronous-requests-in-the-examples>

**Методы массивов**

<https://learn.javascript.ru/array-methods>

Module 4

Creating Forms to Collect and Validate User Input

An HTML5 Form

<form name="userLogin" method="post" action="login.aspx">

<fieldset>

<legend>Enter your log in details:</legend>

<div id="usernameField" class="field">

<input id="uname" name="username" type="text" placeholder="First and Last Name" />

<label for="uname">User's Name:</label>

</div>

<div id="passwordField" class="field">

<input id="pwd" name="password" type="password" placeholder="Password" />

<label for="pwd">User's Password:</label>

</div>

</fieldset>

<input type="submit" value="Send" />

</form>

To change the type of data you wish to collect, specify the type attribute of the input element, providing one of the following values:

* Button
* Checkbox
* Color
* Date
* Datetime
* datetime-local
* email
* file
* hidden
* image
* month
* number
* password
* radio
* range
* reset
* search
* submit
* tel
* text
* time
* url
* week

<input id="ageCategory" name="ageCategory" list="ageRanges" />

<datalist id="ageRanges">

<option value="Under twos"></option>

<option value="2 - 7"></option>

<option value="8 - 12"></option>

<option value="13-17"></option>

<option value="Adult"></option>

</datalist>

<textarea id="carDescription" name="carDescription" cols="80" rows="5"

placeholder="Enter a short description of your car" maxlength="399"/>

<select id="carManufacturer" name="carManufacturer">

<optgroup label="European">

<option value="volvo">Volvo</option>

<option value="audi">Audi</option>

</optgroup>

<optgroup label="American">

<option value="chrysler">Chrysler</option>

<option value="ford">Ford</option>

</optgroup>

</select>

<form id="loginForm" action="login.aspx" method="post" autocomplete="on">

Email: <input name="email" type="email" placeholder="Email address"

autofocus="autofocus"/>

Password: <input name="password" type="password" autocomplete="off" />

<input type="submit" />

</form>

<input id="contactNo" name="contactNo"

type="tel" placeholder="Enter your

mobile number" required="required" />

The required attribute works with the input types **text, search, url, tel, email, password, number, checkbox, radio,** and **file**, and with the input types that pick dates where they are implemented.

<input id="orderRef" name="orderReference" type="text" pattern="[0-9]{2}[A-Z]{3}"

title="2 digits and 3 uppercase letters" />

You can provide feedback to the user about the expected format of the data by using the **title** attribute.

The pattern attribute can be used with the input types **text, search, url, tel, email,** and **password**.

<form id="registerForm" method="post" action="registration.aspx">

<div id="firstNameField" class="field">

<label for="firstName">First name:</label>

<input id="firstName" name="firstName" required="required" placeholder="Your

first name" />

<span style="color:red">\*</span>

</div>

...

</form>

Input

{ border: solid 1px #888; }

input:required

{ border-color: #f00; }

input{ border: solid 1px; }

input:invalid { border-color: #f00; }

input:valid { border-color: #0f0; }

<form id="registrationForm" method="post" action="registration.aspx"

onsubmit="return validateForm();" >

...

<input type="submit" />

</form>

function checkAge() {

// Validate ageInput.value and confirm that the user has specified an age

// in the range 18 to 120 inclusive

let ageValid = ...;

if (!ageValid) {

ageInput.setCustomValidity("Age is invalid. Please specify a value between 18 and

120");

} else {

ageInput.setCustomValidity("");

}

}

...

const ageInput = document.getElementById("confirm-age");

ageInput.addEventListener("input", checkAge, false);

**Note**: You can also use the oninput attribute of an input field to catch the input event,

rather than using the addEventListener function.

<form id="scoreForm" method="post" action="..." onsubmit="return validateForm();" >

...

<div id="scoreField" class="field" >

<label for="score">Score:</label>

<input id="score" name="score" type="number" />

</div>

...

</form>

function isAnInteger( text ){

const intTestRegex = /^\s\*(\+|-)?\d+\s\*$/;

return String(text).search(intTestRegex) != -1;

}

function validateForm()

{

if( ! isAnInteger(document.getElementById('score').value))

return false; /\* No, it’s not a number! Form validation fails \*/

return true;

}

<form id="scoreForm" method="post" action="..." onsubmit="return validateForm();" >

...

<div id="penaltiesField" class="field" >

<label for="penalties">Penalties:</label>

<input id="penalties" name="penalties" type="text" />

</div>

...

</form>

The following JavaScript code performs the validation:

function isSignificant( text ){

const notWhitespaceTestRegex = /[^\s]{1,}/;

return String(text).search(notWhitespaceTestRegex) != -1;

}

function validateForm() {

if( ! isSignificant(document.getElementById('penalties').value))

return false; /\* No! Form validation fails \*/

return true;

}

**Note:** The pattern "\s" in a regular expression matches any whitespace character, so the pattern "[^\s]" matches any characters that are not whitespace. The expression "{1, }" applies the preceding pattern one or more times.

.validatedFine {

border-color: #0f0;

}

.validationError {

border-color: #f00;

}

function validateForm() {

const textbox = document.getElementById("penalties");

if( ! isSignificant(textBox.value)) {

textbox.className = "validationError";

return false; /\* No! Form validation fails \*/

}

textbox.className = "validatedFine";

return true;

}