MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
NATIONAL TECHNICAL UNIVERSITY
"KHARKIV POLYTECHNIC INSTITUTE"

# GUIDELINES

for the laboratory work for the course
"Fundamentals of Web Development"
for students of specialties 121 "Software engineering" and
122 "Computer Science"

APPROVED
editorial and publishing
university council,
Protocol No. 1 of 13.02.2025

Kharkiv
NTU “KhPI”
2025

Guidelines for the laboratory work for the course "Fundamentals of Web Development" for students of specialtys 121 "Software engineering" and 122 "Computer Science", // Author: U.S. Litvinova. – Kharkiv: NTU “KhPI”, 2025. – 56 с.

Author:
U. S. Litvinova

Reviewers:
V.V. Moskalenko

Department of Software Engineering and Management Intelligent Technologies

**CONTENTS**

Laboratory work 1.Topic: Development of web pages with using the HTML language.....4

Laboratory work 2.Topic: Cascading style sheets. Practical use of CSS.....19

Laboratory work 3. Topic: Working with web forms.....31

Laboratory work 4. Topic: Developing dynamic web pages using JavaScript.....40

Laboratory work 5. Topic: Developing dynamic web pages using JavaScript and DOM API.....47

References……60

## Laboratory work 1.

## Topic: Development of web pages with using the HTML language

**The purpose of the work:** In this lab, you will learn the basic techniques of using HTML to create a web page.

**Tasks for the work**

1. Design the structure of a website in the chosen subject area (4-5 pages in length). Consider the placement of navigation bars and duplication of navigation elements. The bottom of each page should contain information about the authors and copyright.
2. Develop a sketch of the website design (use any graphic editor).
3. Create a page layout with a block structure based on the developed sketch.
4. The website should contain the following structural elements of HTML code:

- lists,
- tables,
- images,
- hyperlinks to pages and hyperlinks to positions within a page;
- changed colour/size/font/background colour of the text.

**Only HTML should be used as a web technology in the process of developing a website for this work!**

### Guidelines for the work

**Basic concepts**

**The structure of a web page.**

All web pages consist of two sections - the header (<HEAD>) and the body of the document (<BODY>). The header section can contain text and tags, but the content of this section is not displayed directly on the page.

***Example 1.1*** The simplest HTML document

```
<!DOCTYPE HTML >
<html>
```

<head>
<!-- This section is for the page title and technical information. -->
</head>

<body>
<!—And here you need to place everything that you want to see on the page. -->
</body>
</html>

**Basic elements of a web page.**

*DOCTYPE*

Element <! DOCTYPE> is intended to indicate the type of the current document - DTD (document type definition). This is necessary so that the browser understands how to interpret the current web page, because HTML exists in several versions, in addition, there is XHTML (EXtensible HyperText Markup Language), which is similar to HTML, but differs from it in syntax; so that the browser "does not get confused" and understands according to which standard to display the web page and how it is necessary to specify <! DOCTYPE>.

*Document header section (<HEAD>)*

Tags and text in this section are not displayed on the web page. This section is generally intended for the following service information.

*Page title (<TITLE> tag)*

Used to display a string of text in the title of a browser window bookmark. Such a line tells the user the name of the site and other information that the developer adds.

*CSS (Cascading Style Sheets)*

Styles store a set of formatting elements that are applied to the text of a document to quickly change its appearance.

*Meta tags (<META> tag)*

Meta tags are used to store information intended for browsers and search engines. For example, search engines refer to meta tags to obtain site descriptions, keywords, and other data. Although there is only one <META> tag, it has many parameters. For the visitor of the web page, the information carried by the meta tags will not be visible.

A document can contain any number of <meta> tags. They are all placed in the <head>...</head> block.

Let's consider some commonly used meta tags:

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

It is used so that the browser can correctly determine the type and content and encoding of the web page.

<meta http-equiv="Refresh" content="N; url=http://example.org/">

Automatic redirection (redirect) N seconds after opening the current page to the specified address.

<meta name="author" content="Name of the author of the page">

Used to indicate the author's name. Search engines can find the necessary information on the name of the author.

<meta name="keywords" content="keyword list">

Keywords present in the document are indicated in the keywords meta-thesis. This tag was originally targeted at search engines, but was compromised by webmasters who used it for search spam.

<meta name="description" content="A short description of the page is entered here">

This tag defines the phrase by which the user determines the essence of your page and decides whether to visit it. The expressions entered in this meta-tag play an important role in the ranking of the page. Key phrases from the description should match the main text of the page, this also plays a big role in the indexing of the page by search engines.

<meta name="robots" content="index,all">

Controlling the crawler, telling it that the page should be indexed (or not, if "noindex" is specified).

*Scripts*

A script is traditionally a program that is implemented in the body of a web page and performs certain actions on it. A common programming language for writing scripts is JavaScript.

The order of tags in the title of the document is of no fundamental importance.

Document body (<BODY>)

The body of the document is designed to display data on the web page, in particular, the body contains text, images, links, tables, lists, etc.

*Comments*

Certain text can be hidden from display in the browser by making it a comment. Although the user will not see such text, it will be transmitted in the document, therefore, by looking at the source code, you can find hidden

information. Comments start with the <!-- tag and end with the --> tag. Anything between these tags will not be displayed on the web page.

**Working with text**

*Text formatting*

Formatting text is the means of changing it, such as choosing a font and using effects that allow you to change the appearance of text. Table 1.1 lists the main tags that are used to change the design of the text.

Table 1. 1 - Tags for text formatting

| Code HTML | Description | Example |
|---|---|---|
| Text | Bold text | Text |
|  Text  | Text in italics | Text |
| <sup> Text </sup> | Superscript | $e=mc^2$ |
| <sub> Text </sub> | Subscript | $H_2O$ |
| <pre> Text </pre> | The text is written as is, including all spaces | Text |
|  Text  | Text accentuation | Text |
|  Text  | Important text | Text |

It should be noted that the <B> and <STRONG> tags, as well as <I> and <EM>, are not completely equivalent and interchangeable. The first <B> tag is a physical markup tag and sets bold text, while the <STRONG> tag is a logical markup tag and determines the importance of the highlighted text. This division of tags into logical and physical formatting was originally intended to make HTML universal, including independent of the display device.

Any text formatting tags can be shared.

**Text size**

There are several options for changing the size of the text - the use of <H1>,...,<H6> headings, <BIG> and <SMALL> tags. In the table 1.2 lists the main options with a description and an example.

The <BIG> and <SMALL> tags can be repeated several times in a row, thereby increasing or decreasing the text to the desired size.

Among those presented in the table. 1.1 tags are mainly used tags <H1>, <H2> and <H3>. They are intended to create headings for sections and show their relative importance. Yes, by default, the text inside the <H1> tag is displayed in bold and 24-point size. The content of the <H2> tag is already 18 points, and the <H3> is 14 points.

Table 1.2 - Tags for resizing text

| Code HTML | Description | Example |
|---|---|---|
| <big> Text </big> | Increases the font size | Text |
| <small> Text </small> | Decreases the font size | Text |
| <h1> Text </h1> | Writes the text in the form of a large heading | Text |
| <h6> Text </h6> | Writes the text in the form of a small title | Text |

*Text alignment*

Text alignment determines its appearance, the orientation of the paragraph edges and can be performed on the left or right edge, in the center or in width.

The most common option is alignment on the left edge, when the text on the left is shifted to the edge, and the text on the right remains uneven. Right and center alignment are mainly used in headings and body content. It should be borne in mind that when using width alignment in the text, large intervals may appear between words, which is not very good.

To set text alignment, the paragraph tag <P> is usually used with the align parameter, which specifies the alignment method. It is also permissible to align a block of text using the <DIV> tag with a similar parameter align.,

Alignment of elements on the left edge is set by default, so there is no need to specify it again. Then the align = "left" parameter can be omitted.

The difference between a paragraph (the <P> tag) and the <DIV> tag is that there is vertical indentation at the beginning and end of the paragraph, which is not the case with the <DIV> tag.

The align parameter is quite universal and can be applied not only to the main text, but also to headings, for example <H1>.

**How to work with images**

Inserting images. To embed an image in a document, you use the <IMG> tag, which has the mandatory src parameter that specifies the address of the file with the image and the alt parameter that specifies the alternative text. The general syntax for adding an image is as follows:

<img src='URL' alt=' alternative text'>.

You don't need to close the tag, the URL (Universal Resource Locator) is the path to the image file. You can use either an absolute or relative address to point to it.

As a rule, GIF and JPEG are the most common image file formats.

**Features of GIF**

An image can have from 2 to 256 colours, but they can be any colour from a 24-bit palette.

A GIF file can contain transparent areas. If a background other than white is used, it will show through the 'holes' in the image.

Supports frame-by-frame image changes, which makes the format popular for creating banners and simple animations.

It uses a lossless compression method.

The scope of GIF application

Text, logos, illustrations with sharp edges, animated drawings, images with transparent areas, banners**.**

**Features of JPEG**

The number of colours in an image is about 16 million, which is quite enough to preserve the photographic quality of the image.

The main characteristic of the format is quality, which allows you to control the final file size.

It supports the so-called progressive JPEG technology, in which a low-resolution version of the image appears in the preview window until the image itself is fully loaded.

*Areas of application for JPEG*

Primarily used for photographs. You should not use it for drawings that contain transparent areas, small details, or text.

To change the size of an image using HTML, use the width and height parameters of the <IMG> tag.

It is essential to set the size of all images on a web page. This makes the page load somewhat faster because the browser does not have to calculate the size of each image after it is received.

You can put an image on a web page in a frame of different widths. For this purpose, use the border parameter of the <IMG> tag. By default, the border around an image is not displayed, unless the image is a link. The colour of the frame in this case is the same as the colour of the text specified by the style or the text parameter of the <BODY> tag.

Alternative text allows you to get textual information about an image when image loading is disabled in the browser.

To create alternative text, use the alt parameter of the <IMG> tag.

For images, you can specify their position relative to the text or other images on the web page. The way images are aligned is specified by the align parameter of the <IMG> tag. This parameter can take the following values: bottom ('bottom' aligns the image to the baseline of the text line; this value is set by default), left ('left' aligns the image to the left edge of the parent element), middle ('middle' aligns the image to the baseline of the current text line), right ('right' aligns the image to the right edge of the parent element), top ('top' aligns the image to the topmost element of the current line).

To prevent the text from fitting too closely to the image, it is recommended that you add the hspace and vspace parameters to the <IMG> tag, which set the distance to the text in pixels.

**Links.**

To create a link, you need to tell the browser what the link is and specify the address of the document to which you want to link. Both actions are performed using the <A> tag, which has a single mandatory parameter, href. The value is the document address (URL).

The URL can be absolute or relative. Absolute addresses work everywhere and anywhere, regardless of the name of the site or web page where the link is written. They start with the indication of the data transfer protocol. For example, for web pages, this is usually HTTP (HyperText Transfer Protocol), so absolute links start with the keyword http://

**Work with lists**

*Markers lists*

To set up a bulleted list, use the <UL> and <LI> tags.

Markers lists allow you to divide a large text into separate blocks. This helps to draw the reader's attention to the text and increase its readability. Considering that the perception of text from the monitor screen is more difficult than from its printed version, this is a very useful technique.

Markers can take one of three shapes: circle (default), circle, and square. To select the type of marker, use the type = '...' parameter of the <UL> tag. In place of the three dots, one of the three values specified in Table 1.3 is substituted.

Table 1.3 - Types of Markers

| Code HTML | Example |
|---|---|
| <ul type="disc"> | What to consider when testing a website:<br>– the operability of all links;<br>– support for different browsers;<br>– readability of the text |
| <ul type="circle"> | What to consider when testing a website:<br>– the operability of all links;<br>– support for different browsers;<br>– readability of the text |
| <ul type="square"> | What to consider when testing a website:<br>– the operability of all links;<br>– support for different browsers;<br>– readability of the text |

*Numbered lists*

A numbered list is a set of items with their sequential numbers. The type of numbering depends on the parameters of the <OL> tag used to create the list. As markers, the following values can be used: Arabic numerals; uppercase Latin letters; lowercase Latin letters; uppercase Roman numerals; lowercase Roman numerals.

Before and after the list, vertical indents are automatically added, which is a feature of the <OL> tag.

Table 1.4 - Options for numbered lists

| Code HTML | Example |
|---|---|
| <ol><br><li>текст</li><br><li>текст</li><br><li>текст</li><br></ol> | A numbered list with default parameters:<br>1.;<br>2. text;<br>3. text |

| | |
|---|---|
| <ol start="5"> | A numbered list that starts with five:<br>5. text;<br>6. text;<br>7. text |
| <ol type="A"> | A numbered list with capital letters of the Latin alphabet::<br>A. text;<br>B. text;<br>C. text |
| <ol type="a’> | Numbered list with lowercase Latin letters:<br>a. text;<br>b. text;<br>c. text |
| <ol type="I’> | Numbered list with large Roman numerals:<br>I. text;<br>II. text;<br>III. text |
| <ol type="1"> | Numbered list with Arabic numerals:<br>1. text;<br>2. text;<br>3. text |
| <ol type="I" start="7"> | A list with Roman numerals starting with seven:<br>VII. text;<br>VIII. text;<br>IX. text |

**Tables**

A table consists of rows and columns of cells that can contain text and pictures. Tables are typically used to organise and present data, but they can do more than that. Tables can be used to create page layouts by arranging text and image fragments in the desired way.

To add a table to a web page, use the <TABLE> container tag. A table must contain at least one row and one column (Example 1.1).

To add rows, use the <TR> tag. To divide rows into columns, use the <TD> and <TH> tags.

**Example 1.2** Creating a simple table<!DOCTYPE HTML>

```
<html>
<head>
<meta charset="utf-8">
<title> Table </title>
</head>
<body>
<table>
<tr>
<td> Table contents </td>
</tr>
</table>
</body>
</html>
```

The difference between these tags is as follows. The <TH> tag is intended for creating headings, the content of such a cell is marked with a bold outline and aligned in the centre.

*Features of tables*

Each table parameter has its own default value. This means that if an attribute is omitted, it is implicitly present anyway, and with a certain value, which may result in a different table appearance than the developer intended. To understand what you can expect from tables, you should know their explicit and implicit features, which are described in the following text.

One table can be placed inside the cells of another table. This is necessary for presenting complex data or when one table acts as a modular grid, and the second table inside it acts as a regular table.

Table dimensions are not initially set and are calculated based on the contents of the cells. For example, the total width is determined automatically based on the total width of the cell content plus the width of the borders between cells, the margin around the content set through the cellpadding parameter, and the distance between cells determined by the cellspacing value.

The table is always left-aligned unless otherwise specified. By default, the table is displayed without a border. If you don't specify a table width, it is adjusted to fit the content of the cells.

It is possible that there are extra empty spaces between table cells. This is due to the fact that line breaks in HTML code automatically create an

additional space in the table. To get rid of this, you need to place the code inside the <TR> tag on a single line.

To change the appearance and properties of a table, you can use many parameters that are added to the <TABLE> tag. The general syntax is as follows:

<table parameter1='...' parameter2='...'>

The parameters that are used only for the <TH> and <TD> tags are shown in Table 1.5.

Table 1.5 - Properties of table cells

| Properties | Value | Description | Example |
|---|---|---|---|
| nowrap | | Disables line breaks in the text | <td nowrap> |
| colspan | n | Number of columns to combine | <td colspan="3"> |
| rowspan | n | Number of rows to merge | <td rowspan="3"> |

A description of the table parameters and their values is given in Table 1.6.

Table 1.6 - Parameters of the <TABLE> tag

| Properties | Value | Description | Example |
|---|---|---|---|
| align | left<br>right<br>center | Align the table | <table align="center"> |
| background | URL | Defines the image that will be used as the table background | <table background="pic.gif"> |
| bgcolor | #rrggbb | Background colour of the table | <table bgcolor="#ff9900"> |
| border | n | Frame width in pixels | <table border="2"> |
| cellpadding | n | Distance between a cell and its contents | <table cellpadding="7"> |

| Properties | Value | Description | Example |
|---|---|---|---|
| cellspacing | n | Distance between cells | <table cellspacing="3"> |
| cols | n | Specifies the number of columns in a table, helping the browser prepare for its display | <table cols="3"> |
| nowrap | | Disables line breaks in texti | <table nowrap> |
| frame | void<br>above<br>below lhs<br>rhs<br>hsides<br>vsides<br>box | Setting the type of table border | <table frame="hsides"> |
| rules | all<br>groups<br>cols<br>none<br>rows | Defines where to draw between cells | <table rules="cols"> |
| width | n<br>n% | Minimum table width, can be set in pixels or percentage | <table width="90%"> |

Thanks to a large number of cell properties, you can change the appearance and design of tables.

By default, cell contents are aligned horizontally to the left and vertically to the centre.

The parameters of the <TD> tag take precedence over the parameters of the <TR> tag, and the properties of cells take precedence over the properties of the table itself.

Internet Explorer may not apply some of the parameters attached to the <TR> tag. In this case, you should use the same arguments, but to the <TD> or <TH> tag.

**Special characters**

Table 1 .8 lists some HTML special characters that have a special purpose and a specific way of representing them as mnemonic or numeric code.

Table 1.8 - HTML special characters

| Symbol | Memocode | Numeric code | Description |
|---|---|---|---|
| | | | unbroken space |
| ¢ | ¢ | ¢ | cent |
| £ | £ | £ | pound sterling |
| ¥ | ¥ | ¥ | yen or yuan |
| § | § | § | paragraph |
| © | © | © | copyright mark |
| « | « | « | left double angle bracket |
| | - | - | place of possible transfer |
| ® | ® | ® | registered trademark mark |
| ° | ° | ° | degree |
| 2 | 2 | 2 | uppercase two ($x^2$) |
| 3 | 3 | 3 | uppercase three ($x^3$) |
| · | · | · | point in the middle |
| » | » | » | right double angle bracket |
| ½ | ½ | ½ | fraction - one second |
| × | × | × | multiplication sign |
| ÷ | ÷ | ÷ | division sign |
| σ | Σ | Σ | Greek capital letter sigma |
| λ | λ | λ | Greek lowercase letter lambda |
| μ | μ | μ | Greek lowercase letter mu |
| • | • | • | list marker |

| Symbol | Memocode | Numeric code | Description |
|---|---|---|---|
| … | … | … | ellipsis ... |
| € | € | & # 8364; | euro currency |

**How to document your work:**

1. Title page.
2. Description of the laboratory work.
3. Screenshots and code listing.
4. Conclusions.

**Check questions**

1. What parts does the HTML document consist of?
2. What are the rules for compatibility of HTML syntax with XML?
3. What is the difference between logical and physical formatting of text in an HTML document?
4. How is service information used in the title block of an HTML document?
5. Name the main elements of the page structure.
6. What types of lists can be found in an HTML document?
7. Under what conditions is it better to choose one of the graphic formats GIF or JPG for an image?
8. Why should you always define alternative text for images?

## Laboratory work 2

## Topic: Cascading Style Sheets. Practical use of CSS

The purpose of the work: To study the methods of using style markup. To learn how to create and apply style sheets to control the presentation of web page content.

### Tasks for work

1 To redesign the design of the developed site to use CSS, that is, to develop an external style sheet that can be connected to all pages of the site. The appearance of the pages should be identical. (see task for laboratory work No. 1).

2 To connect the created tables to the web page.

3 To check the correctness of displaying web pages in different browsers.

4 To develop 1 document, where using absolute positioning or floating blocks to determine the information area and the menu area. The bottom of the screen should contain information about the author and copyright.

5 To design the selection of keywords within the text of the developed document using the SPAN tag and assigning the corresponding class in CSS.

### Guidelines for the work

*Cascading Style Sheets (CSS) is a standard that defines the presentation of data in a browser. While HTML provides information about the structure of a document, style sheets tell you how it should look.*

*A style is a set of rules that apply to a hypertext element and determine how it is displayed. A style includes all types of design elements: font, background, text, link colors, margins, and the placement of objects on the page.*

*Cascading is the order in which different styles are applied to a web page. A browser that supports style sheets will apply them sequentially according to their priority: linked styles first, then embedded styles, and finally inline styles. Another aspect of cascading is inheritance, which means that unless otherwise specified, a particular style will be applied to all child elements of a hypertext document. For example, if you apply a specific color to the text in a  tag, all tags inside the block will display the same color.*

*The use of cascading tables makes it possible to separate content and its presentation and flexibly control the display of hypertext documents by changing styles.*

*General CSS syntax*

*Style sheets are built according to a certain order (syntax), otherwise they cannot work properly. Style sheets consist of certain elements (Fig. 2.1):*


Figure 2.1. - CSS style description syntax

Selector. A selector is an element to which the assigned styles will be applied. It can be a tag, class, or a hypertext object identifier of a document.

Property. A property defines one or more characteristics of a selector. Properties define the display format of the selector: padding, fonts, alignment, sizes, etc.

Value. Values are actual numeric or string constants that define the selector property.

Declaration. A collection of properties and their values.

Rule. A complete description of the style (selector + description).

Thus, a style sheet is a set of rules that define the values of the selector properties listed in this table. The general syntax for describing a rule looks like this:

selector[, selector[, ...]] {property: value;}

The character case is not important, the order of listing selectors in the table and properties in the definition is not regulated.

**CSS rules.**

So, a cascading style sheet is a set of rules for formatting HTML tags. Here are some examples of writing such rules:

1. Main text with width alignment, paragraph indentation 30px, typeface (font) – Serif, size (font size) – 14px:p {

text-align: justify;

```
        text-indent: 30px;
        font-family: Serif;
        font-size: 14px;
}
```

This rule will be applied to all tags.

1. Blue color for headings from the first to the third level:

```
h1, h2, h3 {
color: blue; /*same as #0000FF */
}
```

2. Display tables and images without a border:

```
table, img {border: none;}
```

3. Display links in list items without an underline:

```
li a {text-decoration: none;}
```

4. Set the left and right padding for blocks ( ), table headers, and table cells to 10px and fill the background with yellow:

```
div, th, td {
padding-left: 10px;
        padding-right: 10px;
        background-color: yellow;
}
```

1. Display all links in the document in black and bold, and in the main text and lists - in regular font, and also highlight them in green and underline them only when the cursor is hovered over.

```
a {color: black; font-weight: bold;}
pa, li a {font-weight: normal; text-decoration: none;}
pa:hover*, li a:hover {
        color: #00FF00; text-decoration: underline;
        }
```

** - The rule description uses a pseudo-element (a:hover) - an element that is not part of the tag tree structure, but has its own methods and properties.*

**Style classes**

The CSS standard provides the ability to create named styles - style classes. This allows you to answer the following question, for example: How to apply different styles to the same selector?

Suppose that in a document you need two different types of main text - one without indentation, the second - with left indentation and a red font. To do this, you need to create rules for each of them, for example:

```
p {margin-left: 0;}
p.warn {margin-left: 40px; color: #FF00;}
```

To apply the created class, its name must be specified in the class attribute for the selected paragraphs:

```
<p class=”warn”>Red text with left indentation</p>
```

General syntax for describing a class:

```
selector.class_name {description}
```

When creating a class, you can omit the selector, then this rule can be applied to any selector that supports the same set of properties.

***Here are some examples:***
***Rule:***.solid_blue {color: blue;}
**Using:**
 blue paragraph text 
<li class=”solid_blue”> Blue list text </li>
***Rule*:**
h1.bigsans {font-family:Sans; font-size: 1.5em;}
h1.smallserif {font-family:Serif; font-size: .84em;}
**Using:**
<h1 class=”bigsans”> Big but chopped </h1>
<h1 class=”smallserif”> Small but serif </h1>

**Identifiers**

The selector can be the identifier of a hypertext element, specified in the id attribute. To assign styles to such elements, a syntax similar to the description of classes is used, but instead of a period, the # sign (“hash”) is used.

For example:

```
div#content {
        position: absolute;
        top: 10px;
        left: 10%;
        right: 10%;
        border: solid 1px silver;
        }
        ...

<div id="content"> Text </div>
```

It should be remembered that element identifiers must be unique within the document.

**The grouping of properties**

Grouping (grouping) is a combination of values of family qualities. In this case, the style sheet becomes more compact, but stricter requirements are imposed on the description of the rules. Below is an example of a regular style that sets indentations:

```
div {
        margin-left: 10px;
        margin-top: 5px;
        margin-right: 40px;
        margin-bottom: 15px;
}
```

This rule can be rewritten with grouping as follows:

```
div {margin: 5px 40px 15px 10px;} /*order: top right bottom left*/
```

Both styles will be displayed the same.

Grouping can be applied to properties such as margin, padding, font, border, background, etc.

**Linking styles**

There are three ways to apply a style sheet to an HTML document:

– Inline. This method allows you to apply a style to a given HTML tag.

– Embedded. Using it allows you to control the page's styles completely.

– Linked (or External). A linked style sheet allows you to export the style description to an external file, which you can use to control the display of all pages on your site.

***Inline styles***

***Inline styles provide maximum control over all elements of a web page.***

An inline style is applied to any HTML tag using the style attribute, like this:

<p style="font: 12pt Courier">This is 12-point text in Courier font</P>

Example:

<div style="font-family: Garamond; font-size: 18 pt;>"

All text in this section is 18px and in Garamond font.

And this section is also highlighted in red.

Inline styles are useful when you need to fine-tune the appearance of a specific page element or a small web page.

**Inline styles**

Inline styles use the <style> tag, which is placed in the head of an HTML document (<head>...</head>):<html>

<head>

...

<style>

Rules CSS

</style>

```
        ...
</head>
<body>
...
```

**Example of using embedded styles from LR No. 1.**

Linked style sheets.

Linked, or external, style sheets are the most convenient solution when it comes to designing an entire site. The description of the rules is contained in a separate file (usually, but not necessarily, with the .css extension). Using the <link> tag, this style sheet is linked to each page where it needs to be applied, for example:

```
<link rel=stylesheet href="sample.css" type="text/css">
```

Any page containing such a link will be designed according to the styles specified in the sample.css file. It should be noted that the style sheet may physically be located on another web server, then the href must specify the absolute path to it.

**Cascading.**

If you need a hundred or two or three HTML pages, use an external, global style sheet. If some of these pages require general design adjustments, use an embedded style. And if you need to explicitly change the design of one or two elements on the page, use inline styles. This is the order in which styles overlap when cascading, schematically it can be done like this: linked styles -> embedded styles -> inline styles

**Browser issues**

Be sure to view pages with style sheets in different browsers. This is because different browsers may interpret the same rule differently, and some properties and/or values may not be supported at all. You should also test pages with styles disabled (for example, text browsers) to make sure that the page is readable.

**Device-dependent styles**

Style sheets can be used to control the display of content depending on the output device (monitor, projector, printer, sound synthesizer, etc.). To do this, include the device type in the style description, for example:

```
@media print {/* printing device */
        BODY { font-size: 10pt; }
        }
```

```
@media screen { /* monitor */
        BODY { font-size: 12pt; }
        }

@media screen, print {
        BODY { line-height: 1.2; }
        }
@media all {
        BODY { margin: 1pt; }
        }
```

As can be seen from the example, the entire table is divided into sections, each of which begins with the word @media, followed by the name of the device class and then, in curly brackets, the description of the styles themselves.

You can divide style sheets differently by specifying the device type in the tag <link>:

<link rel=stylesheet href="sample.css" type="text/css" media=”screen”>

**CSS Properties**

Table 2.1 lists some commonly used CSS properties and their purpose.

Table 2.1 - CSS Properties

| Name | Value | Description |
|---|---|---|
| background | [background-color \|\| background-image \|\| background-repeat \|\| background-attachment \|\| background-position] \| inherit | Background control element |
| background-color | <color> \| transparent \| inherit | Background color |
| background-image | <uri> \| none \| inherit | Background image |
| background-position | [ [<%] \| <length> ]{1,2} \| [[top \| центр \| bottom] \|\| [Left \| центр \| right] ] ] \| inherit | Background image position |
| background-repeat | Repeat \| repeat-x \| Repeat-y \| no-repeat \| inherit | Repeat background image |

| Name | Value | Description |
|---|---|---|
| border | [border-width \|\| border-style \|\| <color>] \| inherit | Element borders |
| border-collapse | collapse \| separate \| inherit | Merging/dividing adjacent borders |
| border-color | <color>{1,4} \| transparent \| inherit | Border color |
| border-style | <border-style> {1,4} \| inherit | Border line style |
| border-top border-right border-bottom border-left | [ Border-top-width \|\| border-style \|\| <color>] \| inherit | Managing the style of a given border |
| border-width | <border-width>{1,4} \| inherit | Border line thickness |
| bottom | <length> \| < rate > \| auto \| inherit | Bottom of the element |
| clear | none \| left \| right \| both \| inherit | Preventing filling of free space next to an element |
| clip | <Shape> \| auto \| inherit | Trimming the content of an element |
| color | <color> \| inherit | Content color |
| cursor | [ [<uri> ,]* [ auto \| crosshair \| default \| pointer \| move \| e-resize \| ne-resize \| nw-resize \| n-resize \| se-resize \| sw-resize \| s-resize \| w-resize\| Text \| wait \| help]] \| inherit | Cursor shape |
| display | inline \| block \| List-item \| run-in \| compact \| marker \| table \| inline-table \| table-row-group \| table-header-group \| table-footer-group \| table-row \| table-column-group \| table-column \| table-cell \| table-caption \| none \| inherit | How the element is displayed |
| empty-cells | шоу \| hide \| inherit | Displaying empty table cells |
| float | left \| right \| none \| inherit | Free placement of an element |
| Name | Value | Description |

| | | |
|---|---|---|
| font | [[ font-style \|\| font-variant \|\| font-weight]? font-size [/line-height]? font-family] \| caption \| icon \| menu \| message-box small-caption \| status-bar \| inherit | Font management |
| font-family | [[<family-name> \| <generic-family>],]* [<family-name> \| <generic-family>] \| inherit | Text font |
| font-size | <absolute-size> \| <relative-size> \| <length> \| < rate > \| inherit | Kegel |
| font-style | normal \| italic \| oblique \| inherit | Font style |
| font-variant | normal \| small-caps \| inherit | Font display options |
| font-weight | normal \| bold \| border \| lighter \| 100 \| 200 \| 300 \| 400 \| 500 \| 600 \| 700 \| 800 \| 900 \| inherit | Font thickness |
| height | <length> \| < rate > \| auto \| inherit | Element width |
| left | <length> \| < rate > \| auto \| inherit | Position of the element's left border |
| line-height | normal \| <number> \| <length> \| < rate > \| inherit | Row height |
| list-style | [list-style-type \|\| list-style-position \|\| list-style-image] \| inherit | List style |
| margin | <margin-width>{1,4} \| inherit | External indentation |
| margin-top<br>margin-right<br>margin-bottom<br>margin-left | <margin-width> \| inherit | External indentation on a given side |
| padding | <padding-width>{1,4} \| inherit | Inner indentation |
| padding-top<br>padding-right<br>padding-bottom<br>padding-left | <padding-width> \| inherit | Internal indentation on a given side |
| position | static \| relative \| absolute \| fixed \| inherit | Element positioning |
| right | <length> \| < rate > \| auto \| inherit | Right border position |
| Name | Value | Description |

| | | |
|---|---|---|
| text-align | left \| right \| center \| Justify \| <string> \| inherit | Aligning a text block |
| text-decoration | none \| [ underline \|\| overline \|\| line-through \|\| blink] \| inherit | Text effects |
| text-indent | <length> \| < rate > \| inherit | Paragraph indentation |
| text-transform | capitalize \| uppercase \| lowercase \| none \| inherit | Text outline |
| top | <length> \| < rate > \| auto \| inherit | Position of the top border of the element |
| vertical-align | Baseline \| sub \| Super \| top \| text-top \| middle \| bottom \| text-bottom \| < rate > \| <length> \| inherit | Vertical alignment within a block |
| visibility | visible \| hidden \| collapse \| inherit | Control element visibility |
| white-space | normal \| pre \| nowrap \| inherit | Managing spaces between words |
| width | <length> \| < rate > \| auto \| inherit | Element width |
| z-index | auto \| <integer> \| inherit | How to move to the Tab key |

**Positioning elements**

The position property, in combination with the left, top, right, bottom, display, clear and other properties, allows you to control the position of elements on the page and the order in which they are displayed. The position property can take the following values:

1) static - normal position. This block is a regular block, it is displayed according to general rules. The 'left' and 'top' properties do not apply.

2) relative - relative positioning. The position of the block is calculated according to the normal output flow. Then the block is shifted relative to its normal (static) position.

3) absolute - absolute positioning. The position of the block (possibly the size) is specified using the 'left', 'right', 'top' and 'bottom' properties. They indicate the amount of offset relative to the block container. Absolutely positioned blocks are removed from the normal flow. This means that they affect the placement of subsequent elements of the same level.

4) fixed - fixed position. The position of the block is calculated according to the absolute positioning model, and then it is fixed relative to the viewport or page.

Two ads can be separated from each other using the @media rule, as shown in the example:

@media screen { H1#first { position: fixed; } }

@media print { H1#first { position: static; } }

By controlling positioning, you can place blocks of information on the page in different ways, up to creating effects of overlay, flow, gradient, **etc.**

## Check questions

1. What is the difference between display:none and visibility:hidden?

2. I have a 200px wide image on my web page. How do I wrap it around the right side of the image?

3. How do I move a web page element (e.g., <p>) beyond the visible area of the screen?

## Laboratory work 3

### Topic: Working with Web Forms

In this lab, we will look at the process of creating web forms using HTML.

The purpose of this lab is to teach the basics of working with web forms, such as:

- creating web forms;
- styling forms;
- validating input values;
- setting an input mask.

**Tasks for work:**

- creating a registration page with web forms, making your own form. You will also need to create appropriate styles for the design of the web form and validating input values.

### Guidelines for the work

Figure 3.1 shows a simple registration form.

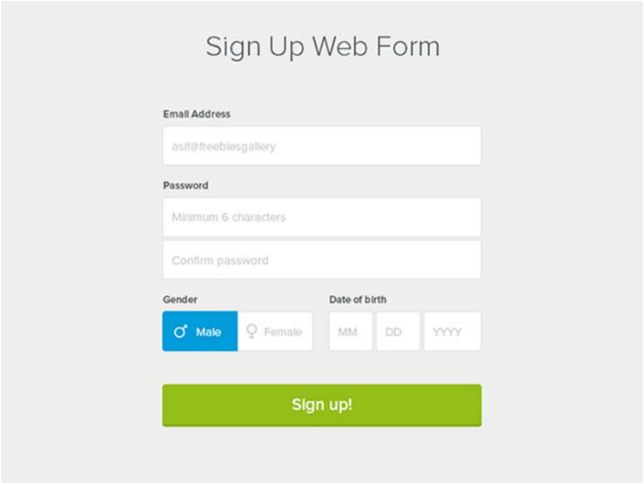


Figure 3.1 - Sample web page layout for task

## 1. Creation of web forms

Let's look at the steps of creating the required page with a registration form.

**Step 1**

Let's create a registration form directly:

```
<form id = "registration">
 <fieldset>
 <legend> Registration Form </legend>
 </fieldset>
</form>
```

Tag <fieldset> is used to logically group shape objects.
Tag <legend> defines a title.

**Step 2**

Add a field for entering the user's full name:

```
<fieldset>
 <legend> Registration Form </legend>

 <label for = fio> Full name </label>
 <input id = fio name = fio type = text>
</fieldset>
```

At this step, the form will look like this in the browser:

**Step 3**

Let's create fields for entering email and phone number, postal code and city:

```
<fieldset>
 <legend> Registration Form </legend>

 <label for = fio> Full name </label>
 <input id = fio name = fio type = text>
```

```
<label for = email> Email </label>
<input id = email name = email type = email>

<label for = phone> Phone number </label>
<input id = phone name = phone type = tel>
</fieldset>
```

As you can see from the following figure, the result is slightly different than what we expected:

**Step 4**

To align form elements line by line and relative to each other, you need to create a table with two columns and no visible borders. Let's change the code as follows:

```
<fieldset>
<legend> Registration Form </legend>
<table class = "alignment">
<tr>
<td> <label for = fio> Full name </label> </td>
<td> <input id = fio name = fio type = text> </td>
</tr>
<tr>
<td> <label for = email> Email </label> </td>
<td> <input id = email name = email type = email> </td>
</tr>
<tr>
<td> <label for = phone> Phone number </label> </td>
<td> <input id = phone name = phone type = tel> </td>
</tr>
</table>
</fieldset>
```

**Step 5**

Let's add a field for entering the date of birth, for this we use type = date (it does not work with all browsers). Let's add the following line to the table:

```
<tr>
 <td> <label for = dateofbirth> Date of birth </label> </td>
 <td> <input id = dateofbirth name = dateofbirth type = "date" /> </td>
 </tr>
```

**Step 6**

For logical grouping, let's add another fieldset that combines the elements indicating the address and postal code:

```
<fieldset>
 <legend> Delivery parameters </legend>
 <table>
 <tr>
 <td> <label for = address> Address </label> </td>
 <td> <textarea id = address name = address rows = 5> </textarea> </td>
 </tr>
 <tr>
 <td> <label for = postcode> Postcode </label> </td>
 <td> <input id = postcode name = postcode type = text> </td>
 </tr>

 </table>
 </fieldset>
```

Since writing the full address will most likely take more than one line, let's set the attributerows, with a value equal to 5 for the corresponding tag.

**Step 7**

We will also select a control in a separate logical group, with the help of which the user will indicate the preferred method of receiving notifications:

```
<fieldset>
 <legend> Preferred way of receiving notifications </legend>
 <table>
 <tr>
 <td> <input id = emailmessage name = message type = radio> </td>
 <td> <label for = emailmessage> By Email </label> </td>
 </tr>
 <tr>
 <td> <input id = phonemessage name = message type = radio> </td>
 <td> <label for = phonemessage> By phone </label> </td>
 </tr>
 <tr>
 <td> <input id = nomessage name = message type = radio> </td>
 <td> <label for = nomessage> Don't notify me </label> </td>
 </tr>
 </table>
 </fieldset>
```

**Step 8**

It remains only to add a button to the page:

```
<fieldset>
 <button type = submit> Submit data </button>
 </fieldset>
```

## Check questions

1. Which attribute can be used to control data entry into a form?
2. What attributes are available in HTML5 for forms?
3. How to protect web forms from spam?
4. What are the tools for developing forms?
5. How to create a simple slide using HTML?

## Laboratory work № 4

### Topic: Developing dynamic web pages using JavaScript

**The purpose of this laboratory work** is to consolidate the theoretical material from the previous lecture material, study the basics of the JavaScript scripting language for further use in developing websites.

**Tasks for work:**

1. To implement the tasks, add a page named "JavaScript.html" to the site created within the framework of laboratory works No. 1,2,3.
2. Create a form with an input field and two buttons (OK and CANCEL). By clicking on the OK button in the form, fill in the text field with information (a) according to the option.
3. Create the file "lr4.js", in which to implement the task (b) according to the option, and connect this file to JavaScript.html.
4. Add a hyperlink, when clicked on which, information (C) will be displayed in the window according to the option.

### TASK OPTIONS

**Option 1**

A. The number of days (calculated in the script) that have passed since your birthday.

B. Write a script that determines the number of days until the nearest Sunday, and display this number when the page loads in a message box.

C. Report the name of the current month.

**Option 2**

A. The number of weeks (calculated in the script) that have passed since your birthday.

B. Write a script that determines how many days have passed since the New Year, and display this number when the page loads in a message box.

C. Report the number of hours and minutes that have passed since the beginning of the day.

**Option 3**

A. The number of hours and minutes that have passed since the beginning of the current month.

B. Write a script that determines the number of hours (Greenwich Mean Time) until the New Year, and display this number when the page loads in a message box.

C. Report the current cursor coordinates.

**Option 4**

A. Full information about the current date and time. For example, "May 14, 2021, Tuesday, 2:53:44 pm".

B. Write a script that determines how many weeks are left until September 1 of the current year.

C. Report the value of the current list item.

**Option 5**

A. The number of weeks that have passed since the last September 1.

B. Write a script that determines whether the current year is a Summer Olympics year (leap year) and displays this number in a message box when the page loads.

C. Report the value of the current list item.

**Option 6**

A. The number of hours (calculated in the script) left until the beginning of summer.

B. Write a script that determines how many days are left before the vernal and autumnal equinoxes (22 March and 22 September) and print this number when the page loads in a message box.

C. A message about the number of times the event handler is triggered in the current minute.

**Option 7**

A. The number of hours and minutes that have passed since the beginning of the current month.

B. Write a script that determines: half-year (first or second); quarter (first, second, third or fourth); season (winter, spring, summer or autumn); century; millennium; and display this information when the page loads in the message box.

C. Show the message of the current second.

**Option 8**

***A***. How many days are left until the next Friday, which falls on the 13th day.

B. Write a script that determines how many days are left until the nearest 23rd of August.

C. Display the current time.

**Option 9**

A. The number of days left before the summer holidays (specify the date of the beginning of the holidays in the dean's office).

B. Write a script that determines the number of hours left until the end of the current month and display this number when the page loads in a message box.

C. Display a random number from 1 to the number of days in the current month.

**Option 10**

A. The number of minutes that have passed since the beginning of the pair.

B. Write a script that determines how many days have passed since the last 14th of February.

C. A message about the number of times the event handler has been triggered in the current session.

**Select options from the group list. After 10, the number will be 1 again.**

## Guidelines for the work

**JavaScript** is a client-side scripting language for viewing hypertext web pages. To be more precise, JavaScript is not only a client-side programming language. Liveware, the ancestor of JavaScript, is a server-side substitution tool from Netscape. However, it is client-side programming that has made JavaScript most popular.

### Placing JavaScript code on an HTML page

In general, there are four ways of using JavaScript functionally:
1) hypertext link (URL scheme);
2) event handler (in attributes corresponding to events);
3) insertion (<SCRIPT> container).

### Comments in HTML and JavaScript

In a JavaScript program, you can leave comments that are ignored by the JavaScript interpreter and serve as an explanation for developers. Single-line comments begin with the // characters. The text starting with these characters and ending with the end of the line is considered a comment. A multi-line

comment is enclosed between the characters/* and */ and can occupy several lines.

**Data types and operators**

Like any other programming language, JavaScript supports built-in structures and data types. All their variety is divided into: literals; variables; arrays; functions; objects.

They are further divided into: built-in and programmer-defined.

***Literals***

***A literal is data that is used directly in a program. Data can be*** understood as numbers or strings of text. All of them are considered elementary data types in JavaScript.

Literals are used in operations of assigning values to variables or in comparison operations:

```
var a = 10;
var str = ‘String’;
if(x==‘test’) alert(x);
```

The assignment operator (variable = expression) returns the result of evaluating an expression, so nothing prevents you from assigning the resulting value to another variable.

In addition to string literals (sequences of characters enclosed in quotes), there are also string objects; they are created by the constructor: var s = new String(). This object has many methods. You should understand that a string literal and a string object are not the same thing.

***Variables.***

A variable is a memory location that has a name and stores certain data. Variables in JavaScript are declared using the var statement, and you can assign them initial values or not:

```
var k;
var h='Hello!
```

The type of a variable is determined by the value assigned to it. JavaScript is a weakly typed language: you can assign different types to the same variable in different parts of the program, and the interpreter will change the variable type on the fly. You can find out the type of a variable using the typeof() operator.

A variable declared with the var statement outside of a function is global - it is 'visible' everywhere in the script. A variable declared with the var operator inside a function is local - it is visible only within that function.

You can also declare variables without the var statement by simply assigning an initial value to the variable. However, it is not recommended to omit the var statement.

***Arrays***

Arrays are divided into built-in arrays (document.links [], document.images [], etc. - they are also called collections) and user-defined arrays (by the document author). There are several methods for arrays: join(), reverse(), sort(), and others, as well as the length property, which allows you to get the number of elements in an array.

To define a user array, there is a special Array constructor. If one argument is passed to it, and it is a non-negative integer, then an empty array of the corresponding length is created. If one argument is passed that is not a number, or more than one argument, then an array is created filled with these elements:

a = new Array(); // empty array (length 0) b = new Array(10); // array of length 10

c = new Array(10,'Hello'); // an array of two elements: numbers and strings

//A short way to create an array of 4 elements d = [5, 'Test', 2.71828, 'Number e'];

Array elements are numbered from zero. Therefore, in the last example, the value of d[0] is 5, and the value of d[1] is 'Test'. Thus, an array can consist of heterogeneous elements. Arrays cannot be multi-dimensional, but you can create an array whose elements are also arrays.

The join() method allows you to combine the elements of an array into a single string. It is the opposite of the split() method discussed above, which cuts an object of type String into pieces and makes an array out of them. By the way, the split() method demonstrates the fact that an array can be obtained without an array constructor.

The reverse() method is used to reverse the order of the array elements. You can reorder it in reverse order by calling the a.reverse() method.

The sort() method interprets array elements as string literals and sorts the array in alphabetical (lexicographic) order. Note that the sort() method changes the array. The sort() method interprets array elements as strings (and

performs lexicographic sorting), but does not convert them to strings. If there were numbers in the array, they will remain numbers.

**Operators of condition**

The main focus is on declaration and flow control statements. No JavaScript program can be written without them.

The list of basic operators is as follows:

{...}

if... else...

()?

while for

break

continue return

The {...} statement

Curly braces define a compound JavaScript block. The main purpose of a block is to define the body of a loop, the body of a conditional statement, or a function.

***The if... else... statement***

A conditional statement is used to branch a program according to a certain logical condition. There are two syntax options:

if (boolean expression) operator_1;

if (logical_expression) statement_1; else statement_2;

A Boolean expression is an expression that takes a value of true or false.

***Operator().***

This operator, called a conditional expression, produces one of two values depending on whether a certain condition is met. Its syntax is as follows:

Boolean_expression)? value_1 : value_2

If the Boolean expression is true, then value_1 is returned, otherwise, value_2 is returned. The conditional expression is easily imitated by the if... else statement, but it allows you to make your program code more compact and easy to read.

The **while** operator defines a loop, which is generally defined as follows:

while (loop_continuation_condition) loop_body;

The **loop** body can be either a simple or a complex operator. A complex operator is usually enclosed in curly braces. It is also recommended to enclose a simple operator in them so that the program can be easily modified. The loop_continuation_condition is a logical expression. The body is executed as long as the logical condition is true.

The for operator is another loop operator. In general, it has the form:

for (loop_variable_initialization; loop_continuation_condition; loop_variable_modification)

loop_body;

The loop body can be either a simple or a complex operator (complex operators must be enclosed in curly braces). The loop_variable_initialization and loop_variable_modification operators can consist of several simple operators, in which case the simple operators must be separated by a comma. The loop_continuation_condition is a logical expression.

The **break** operator allows you to leave the loop body early.

The **continue** operator allows you to proceed to the next iteration of the loop, skipping the execution of all subsequent operators in the loop body.

The **return** operator is used to return a value from a function or event handler.

Note: the return operator not only specifies what value the function should return, but also stops the execution of subsequent operators in the function body. When used in event handlers, the return operator allows you to cancel or not cancel the default action that the browser creates when a given event occurs.

**Check questions**

1. Name the ways to place JavaScript code on an HTML page.
2. How to add comments to JavaScript code?
3. Why is JavaScript a weakly universal language?
4. How to find out the type of a variable in JavaScript?
5. What are literals?
6. What is the var operator used for?
7. Name the features of using the var operator inside functions.
8. What types of arrays exist in JavaScript?
9. Name the methods of the "array" object and the purpose of these methods.
10. What operators do you know in JavaScript?

## Laboratory work 5

## Topic: Developing dynamic web pages using JavaScript and DOM API

**The purpose of the laboratory work** is to consolidate the theoretical material from the previous lecture material, study the document object model and the means of working with it in the JavaScript scripting language when developing dynamic websites.

### Tasks for work:

1. Add the following functionality to the site created as part of the previous laboratory work:

- Add a block in which the current date is inserted when the page is loaded.
- After hovering the mouse over the area with the date, the current time (hours; minutes; seconds) begins to be displayed in it, which is updated every second. After moving the mouse cursor from this area, the current date is displayed in it again.
- Add a logo or another (more appropriate, in your opinion) image implemented by several images to the site. After hovering the mouse cursor over the logo, the parts of the image that make up it should be replaced with brighter ones (more interesting solutions are welcome).
- Develop a function that, when hovering the mouse over the "About the site" hyperlink, should display a block with brief information about the authors below it. This should look like a pop-up window, but without additional windows, and using only HTML blocks.
- Image loading should be optimized.

### Guidelines for the work

To implement interactive, dynamic web pages, the so-called Dynamic HTML, or DHTML, is used.

Dynamic HTML is a way of creating an interactive website that uses a combination of the static HTML markup language, embedded (and executed on the client side) with the JavaScript scripting language, CSS (cascading style sheets), and DOM (document object model).

**Objects in JavaScript**

An object is the main JavaScript data type. The Object data type itself defines objects. An object in JavaScript is a regular associative array ("hash"). It stores any "key => value" correspondences and has several standard methods. Several types of objects can be used in a JavaScript script:

– client-side objects are included in the DOM model, that is, they correspond to what is contained or occurs on the Web page in the browser window. They are created by the browser during parsing of the HTML page. Examples: window, document, location, navigator, etc.;

– server objects are responsible for client-server interaction. Examples: Server, Project, Client, File, etc. Server objects are not considered in this course;

– built-in objects are various data types, properties, methods inherent in the JavaScript language itself, regardless of the content of the HTML page. Examples: built-in object classes Array, String, Date, Number, Function, Boolean, as well as the built-in Math object;

– user objects are created by the programmer in the process of writing a script using object type (class) constructors.

**Operators of working with objects for... in...**

The for operator (in variable of the object) allows you to "run" through the properties of the object.

The with operator sets the default object for the block of operators defined in its body. Its syntax is as follows:

with (object) operator;

The properties and methods inherent in the operator body must be written in full, otherwise they will be considered properties and methods of the object specified in the with operator. The with operator is useful when working with the Math object, which is used to access mathematical functions and constants.

**Client objects**

The Document Object Model (DOM) is used to create a mechanism for managing pages on the client side. The essence of the model is that each HTML container corresponds to an object characterized by a triple: properties; methods; events.

The object model can be imagined as a way of communicating between pages and the browser. The document object model is a representation of objects, their methods, properties and events that are present and occur in the

browser software, in the form of a convenient script for working with them from HTML code and source text on the page. With its help, you can set any requirements for the browser and transfer them to client pages. The browser will execute commands and, accordingly, change the page on the screen.

Objects with the same set of properties, methods and events are combined into classes of objects of the same type. Classes are descriptions of possible objects. The objects themselves appear only after the browser loads the document or as a result of the program. You must always remember this so as not to refer to a non-existent object.

**DOM class hierarchy**

An object-oriented programming language assumes the existence of a hierarchy of object classes. In JavaScript, such a hierarchy begins with the Window object class, that is, each object is assigned to a particular window. To access any object or its properties, specify the full or partial name of this object or its properties, starting with the name of the object that is higher in the hierarchy to which this object belongs

However, JavaScript is not a classical object language (it is also called a lightweight object language). It lacks inheritance and polymorphism. There are only theses "object A contains object B". It is not a class hierarchy in the literal sense. DOM objects necessarily have certain properties, while the presence of others depends on the web page. For example, the Window object always has the location and history objects as its properties, that is, these are mandatory properties. If the HTML page contains a <BODY> container, then the Window object will have the document object as a property. If the HTML page contains a <FRAMESET> container with <FRAME> containers nested within it, then the Window object will have the frame names as properties, for example window.f1. The latter are themselves objects of the Window class, and for them, in turn, all of the above is true.

**Note**. Each browser, be it Internet Explorer, Mozilla Firefox or Opera, has its own object model. The object models of different browsers (and even different versions of the same one) differ from each other, but have essentially the same structure. Therefore, there is no point in dwelling on each of them separately.

**Collections**

A collection is a JavaScript data structure similar to an array. The difference between a collection and an array is that the programmer creates

arrays in the program code and fills them with data. Collections are created by the browser and are "populated" with objects associated with web page elements. A collection can be considered a more convenient way to access web page objects.

**For example**, if there are forms on a page with the names f, g5, and h32, then the document object has the corresponding object properties document.f, document.g5, etc. In addition, the document object has a properties forms, which is a collection (array) of all forms. Therefore, the same form objects can be referred to as document.forms[0], document.forms[1], etc. This is convenient when performing actions with all form objects on a given page. When defining the properties of a particular object, collections will be written with brackets: forms[], images[], frames[], to emphasize that these are not ordinary properties, but collections.

The elements of a collection are numbered, starting from zero, in the order of their appearance in the original HTML file. Collection elements are accessed either by index (in parentheses or square brackets) or by name (also in parentheses or square brackets or through a period). Like ordinary arrays, collections have a length property, which allows you to find out the number of elements in the collection.

**Properties**

Most HTML containers have attributes. As you know, each container corresponds to an object. Therefore, the corresponding attributes correspond to the properties of the object. The correspondence between the attributes of HTML containers and the properties of DOM objects is not always direct.

Usually, each attribute corresponds to a certain property of the object. But, firstly, the name of this property is not always easy to understand from the name of the attribute, and secondly, the object may have properties that have no analogues among the attributes. In addition, attributes are case-sensitive or case-insensitive, like the entire HTML language, while the properties of objects must be written in a precisely defined case of characters. Properties can also be accessed using the parenthetical notation: object ['property']. Objects corresponding to hyperlinks also have properties that have no analogues among the attributes. A complete list of properties of objects of the URL class is located in the JavaScript reference.

**Methods**

In JavaScript terminology, object methods define functions that perform actions on this object, for example: changing its properties, displaying them on a web page, sending data to the server, reloading the page, etc.

**Events**

In addition to methods and properties, objects are characterized by events. Actually, the essence of JavaScript programming is to write handlers for these events. For example, a Click event can occur with an object of type button (INPUT container of type button - "button"), that is, the user can click on the button. For this, the attributes of the INPUT container are extended with the attribute for handling this event - onClick. The value of this attribute is the event handling program that the author of the HTML document must write in JavaScript.

Event handlers are specified in specially created attributes in those containers with which these events are associated. For example, the BODY container defines the properties of the entire document, so the event handler "Completed loading of the entire document" is specified in this container as the value of the onLoad attribute.

**Prototype**

Usually, developers use built-in JavaScript objects (in addition to the DOM object hierarchy), such as: Data, Array, and String. In this sense, the property of objects called prototype is interesting. Prototype is another name for the constructor of an object of a particular class. There is one significant nuance: only those objects that are generated after changing the prototype of the object will have new methods and properties. All built-in objects are created before the JavaScript program receives control, which significantly limits the use of the prototype property.

**Window object**

The Window object class is the oldest class in the JavaScript object hierarchy. The window object, which refers to the current window (i.e., in which the script is executed), is an object of the Window class. The Frame object class is contained in the Window class, that is, each frame is also an object of the Window class.

The window object is created only when the window is opened. All other objects that are created when a page loads are properties of the window object. Moreover, all global variables defined in this window are also properties

of the window object. Thus, the window object can have different properties when loading different pages. In addition, in different browsers, the properties of objects and the behavior of objects and the browser when handling events may be different.

Since the window object is the oldest, in most cases, when referring to its properties and methods, the prefix "window." can be omitted (of course, if you need to refer to a property or method of the current window where the script is running; if it is a different window, then you must specify its identifier). For example, you can write alert ('Hello') instead of window.alert ('Hello'), or location instead of window.location. Exceptions to this rule are calls to the open() and close() methods, in which you must specify the name of the window you are working with (the parent in the first case and the child in the second).

**Methods of the window object**

What operations can you perform with a window? Open (create), close (delete), put it on top of all other open windows (transfer focus). In addition, you can manage the properties of the window and the properties of its subordinate objects. You should focus on the simple and most popular methods of window management.

The alert() method allows you to create a warning window that has only an "OK" button:

<A HREF="javascript:window.alert('Attention')"> Repeat the request!</A>

It should be borne in mind that messages are displayed in the system font, therefore, to receive warnings in Russian, a localized version of the OS is required.

The confirm() method allows the user to ask a question that can be answered positively (by clicking the "OK" button) or negatively (by clicking the "Cancel" button or simply closing the query window). Depending on the user's actions, the confirm() method returns true or false**.**

The prompt() method allows you to accept a string of text from the user.
prompt("Question String", "Default Answer String")

When the user enters their answer (or leaves the default answer unchanged) and clicks the OK button, the prompt() method will return the received string as a value that can be further acquired by any variable and which can be further parsed in a JavaScript program.

The open() method is intended for creating new windows. In the general case, its syntax looks like this:

```
myWin = window.open("URL", "window_name",
"parameter = value, parameter = value,... ", replace);
```

The first argument specifies the address of the page that is loaded into the new window (you can leave the string empty, then the window will remain empty). The second argument specifies the name of the window, which can be used in the TARGET attribute of the <A> and <FORM> containers. The reserved names _blank, _parent, _self, _top are also allowed as values, the meaning of which is the same as that of the similar values of the TARGET attribute. If the window_name matches the name of an already existing window (or frame), then a new window is not created, and all subsequent manipulations with the myWin variable will be applied to this window (or frame).

The third argument does not contain spaces in the lines and is a list of parameters and their values, separated by commas. Specifying each of the parameters is optional, but the default values may depend on the browser, so you should always specify exactly those parameters that are expected.

The window.open() method returns a reference to the newly opened window, that is, an object of the Window class. It can be assigned to a variable (as was done above) so that in the future you can control the open window (write to it, read from it, transfer and remove focus, close it).

The close() method allows you to close the window. The most common question is which window should be closed. If you need to close the current one, then:

```
window.close(); self.close();
```

If a window is opened using the window.open() method, then from a script running in a new window, you can refer to the parent window using window.opener (here window refers to the object of the newly created window, since it is used in the script running in the new window). Therefore, if you need to close the parent window, that is, the window from which the current one was opened, then:

```
window.opener.close();
```

If you need to close a free window, you first need to get its identifier:

```
id=window.open();
...
id.close();
```

As can be seen from the last example, the window is not closed by name (the value of the TARGET attribute is not taken into account), but a pointer to the object is used.

focus() and blur()

The **focus()** method is used to transfer focus to the window with which it was used. Transferring focus is useful both when opening a window and when closing it, not to mention cases when you need to select windows. The following is an example.

Open a window and, without closing it, open a window again with the same name, but with different text. The new window did not appear on top of the main window, because the focus was not transferred to it. Now open the window again, but with the focus transferred. Since the contents of the new window are written from the old (parent) window, the value of the myWin variable is used as a pointer to the object. To take the focus away from a specific myWin window, you need to use the myWin.blur() method.

For example, to take the focus away from the current window where the script is running, you need to call window.blur(). The effect will be as if the user himself minimized the window by clicking the button in the upper right corner of the window.

**The setTimeout()** method is used to create a new calculation thread, the execution of which is postponed for the time (in milliseconds) specified by another argument:

```
idt = setTimeout("JavaScript_code",Time);
```

A typical application of this function is organizing periodic changes in object properties. For example, you can start a clock in a form field.

The clearTimeout() method allows you to destroy the thread called by the setTimeout() method. Its use allows you to more effectively distribute the resources of the computing device.

**Events of the window object**

It is advisable to dwell on the events associated with the window object. Handlers of these events are usually placed as an attribute of the <BODY> container.

**Load** – the event occurs when the document loading in this window is completely completed. If the current window is a frame, then the Load event of its window object occurs when the document loading in this frame is complete, regardless of the state of document loading in other frames.

You can use the handler for this event, for example, as follows:

<BODY onLoad="alert('Document fully loaded.');"> Unload

**Unload** – the event occurs when the page is unloaded from the window. For example, when the user closes the window or moves from this web page to another in the case of a link or typing an address in the address bar or in the case of changing the page address (window.location property) with a script. For example, if the user leaves our page, you can take care of his convenience and close the window previously opened by our script:

<BODY onUnload="myWin.close();">

**Error** – the event occurs when an error occurs during the page loading process. If this event occurs, you can, for example, display a message to the user using alert() or try to reload the page using window.location.reload().

**Focus** – the event occurs when the focus is transferred to the window. For example, when the user "reopens" a previously minimized window or (in Windows) selects this browser window using Alt + Tab among the windows of other applications. This event also occurs when the window data focus is transferred programmatically by calling the window.focus() method.

Example of use:

<BODY onFocus="alert('Thank you for coming back!');">

**Blur** – the opposite event to the previous one, occurs when this window loses focus. This can happen as a result of user actions or by software means – by calling the window.blur() method.

Resize – the event occurs when the window is resized by the user or a script.

**Document Object Model (DOM)**

**Most actions in JavaScript are performed on an HTML page. In** JavaScript, a page is represented as a DOM (Document Object Model) object.

Any actions on a page require calling the appropriate DOM method.

**The document object**

The document object is the most important property of the window object (i.e., it must be fully addressed as window.document). All HTML markup elements present on a web page - text, paragraphs, hyperlinks, images, lists, tables, forms, etc. - are properties of the document object. We can say that DHTML (Dynamic HTML) technology, i.e. dynamic changes in the content of a web page, consists precisely in working with the properties, methods, and events of the document object.

The ability to work with a document in the DOM model is a cornerstone in JavaScript programming.

Access to elements

Any access and changes to the DOM originate from the document object.

The top of the tree

document.documentElement

The topmost tag. In the case of a valid HTML page, this will be <html>. document.body. The <body> tag, if present in the document (required).

**DOM element types**

Each element in the DOM model has a type. Its number is stored in the elem.nodeType attribute. In total, 12 element types are distinguished in the DOM. Usually only one is used: Node.ELEMENT_NODE, whose number is 1. Elements of this type correspond to HTML tags.

Sometimes the Node.TEXT_NODE type is also useful, which is 3. These are text elements.

The remaining types are not used in JavaScript programming.

**Child elements**

From the top of the tree, you can go further down. To do this, each DOM node contains an array of all descendants, separately - references to the first and last descendants and a number of useful properties.

All child elements, including text ones, are in the childNodes array. The firstChild and lastChild properties point to the first and last child elements and are null if there are no descendants.

The parentNode property points to the parent. For example, for <body> such an element is <html>.

The previousSibling and nextSibling properties point to the left and right siblings of the node.

*Element properties*

DOM elements have a lot of properties. Usually, a maximum of a third of them are used. Some of them can be read and set, others can only be read.

There is also a third option, found in IE, when a property can be set only when creating an element.

It is worth considering some (not all) of the element properties that are useful when working with the DOM.

*tagName*

The attribute is present in tag elements and contains the tag name in uppercase, it is read-only.

*style*

Change individual element styles. All HTML elements have a style property. With its help, you can easily change the styles that are specified for the element in the HTML style attribute.

*innerHTML*

This property contains all the HTML code inside the node, and it can be changed. The innerHTML property is mainly used to dynamically change the content of the page.

*className*

This property specifies the class of the element. It is completely analogous to the HTML "class" attribute, for example:

```
elem.className = 'newclass'
```

***Image Object***

The most spectacular effects in JavaScript programming are achieved when working with graphics. At the same time, the programmer does not have many tools in his arsenal: images embedded in the document, the ability to generate an Image object, combining images with hypertext links and tables. However, the number of various effects that are achieved with these simple means is impressive.

Graphics programming in JavaScript is based on the Image object, which is characterized by the following properties, methods and events Despite the significant number of properties, the absolute majority of them can only be read, but not changed. This is evidenced, first of all, by the lack of methods. But two properties can still be changed: src and lowSrc. This turns out to be enough for many effects with images.

*Changing the image*

You can change the image only by assigning a new value to the src property of the embedded Image object. Above, it was shown how this can be done in a simple case. Obviously, slow reloading of the image from the server does not allow for fast scrolling.

The solution is to time-shift the image loading and display. First, an image is created, to which the onMouseOver and onMouseOut event handlers are attached. After hovering the mouse pointer over each of the images, it is replaced by another (colored) one, and when the mouse is moved away, the image becomes black and white again

```
<IMG NAME=m0 src="images/mapb000.gif" border=0
onMouseOver="document.m0.src=color[0].src;"
onMouseOut="document.m0.src= mono[0].src;">
```

*Animation*

A natural extension of the idea of replacing the SRC attribute value in the IMG container is animation, that is, a sequential change in the value of this attribute over time. To implement animation, the setTimeout() method of the window object is used.

There are two ways to start animation: after the page is loaded (onLoad) and during user actions (onClick, onChange, etc.). The most popular is the first one, namely the use of onLoad() and setTimeout().

*The onLoad event handler*

The Load event occurs when the browser has finished loading the document. The handler for this event (onLoad) is specified in the BODY container:

```
<BODY onLoad="JavaScript program">
```

*Display optimization*

When programming graphics, you should take into account many factors that affect the speed of displaying the page and the speed of changing graphic images. At the same time, the usual dilemma of optimizing programs - speed or size of memory occupied - is solved only in favor of increasing speed. The size of memory in JavaScript programming is not taken into account.

Of all the ways to optimize the display of images, you should focus on only a few:

- display optimization during loading;
- display optimization due to preloading;
- display optimization due to image slicing.

If the first two positions apply to both the display of static images and animation, then the third point is characteristic mainly for animation.

**Check questions**

1. What is DHTML?
2. What is DOM?
3. Name the basic structure of the DOM, describe its main objects.
4. What types of objects exist in JavaScript?
5. Name the uses of the this keyword.
6. What is the difference between changing individual element styles and accessing the current element styles?
7. How to find an element by its identifier?
8. How to form a collection of elements of the same tag?
9. How to form a collection of elements of the same class?

## References

1. Wagner G. Building Front-End Web Apps with Plain JavaScript.,2020.-333c.
2. Duckett Jon. HTML and CSS: Design and Build Websites, 2020. - 514 c.
3. Herron D. Node.js Web Development – 4 edition., 2018. – 492c.
4. Mike McGrath. CSS in Easy Steps, 2020.- 192c.
5. Haverbeke M. Eloquent JavaScript 3rd edition., 2018.-436c.

Educational edition

**GUIDELINES**

for the laboratory work for the course
"Fundamentals of Web Development"
for students of specialties 121 "Software engineering" and
122 "Computer Science"

Author:

LITVINOVA Uliya Sergiivna

Responsible for the issue Assoc. Prof. Kopp A.M.
The paper has been recommended for publication by Prof. Hamaiun I.P.

In the author's original version

Plan of 2025, pos. 175

Signed for publication on 13.02.2025.
Headset Times New Roman.
Approx. printed pages 3,5.

Publishing Centre of NTU "KhPI
Certificate of state registration DK № 5478 of 21.08.2017.
61002, Kharkiv, Kyrpychova str, 2

Independent electronic publication