OBJECT MODELS: DOM, BOM AND JS

Boris Bocharov, Maria Voevodina,
Nataliia Braterska, Anastasiia Dashkovska

By itself, the JavaScript language does not provide for working with the browser [1-3].

It does not know about HTML at all. But it allows you to easily expand yourself with new functions and objects.

The figure 1 schematically shows the structure, which is obtained if you look at a set of browser objects from the "bird's eye view".

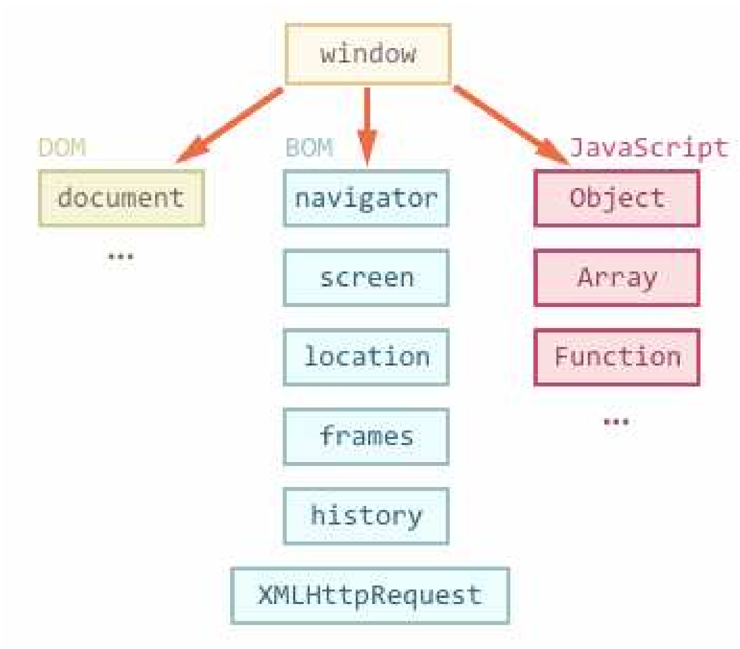

Figure 1. Object models

As can be seen from the figure, the top is window.

This object has a dual position - on the one hand it is a global object in JavaScript, on the other hand it contains properties and methods for controlling the browser window, opening new windows, for example:

```
// open a new window / tab with the URL http://google.ru
window.open ('http://google.ru');
```

The Document Object Model (DOM)

The global document object allows you to interact with the contents of the page. Example of use:

```
document.body.style.background = 'red';
alert ('The BODY element has turned red, and now it's back');
document.body.style.background = '';
```

It and a huge number of its properties and methods are described in the standard W3C DOM.

For historical reasons, the first version of the DOM Level 1 standard once appeared, then came up with more properties and methods, and DOM Level 2 appeared, at the moment, DOM Level 3 is added on top of them and DOM 4 is being prepared.

Modern browsers also support some features that are not included in the standards, but de facto exist a long time ago and nobody wants to abandon them. They are conventionally called "DOM Level 0".

Also information on working with page elements can be found in the HTML 5 standard.

DOM Tree

The main tool of work and dynamic changes on the page is the DOM (Document Object Model) - the object model used for XML / HTML documents.

According to the DOM-model, the document is a hierarchy, a tree. Each HTML tag forms a tree node with an "element" type. Nested tags become child nodes. To represent the text, nodes with the type "text" are created.

The DOM is a document view in the form of an object tree, which is available for modification through JavaScript.

All that is in HTML, is in the DOM.

Even the <! DOCTYPE ...> directive we are putting at the beginning of HTML is also a DOM node, and is in the DOM tree just before <html>. Even the document object itself, formally, is the DOM node, the very root one.

There are 12 types of nodes in total, but in practice we work with four of them:

```
1. The document is the entry point to the DOM.
2. Elements - the main building blocks.
3. Text nodes - contain, in fact, the text.
4. Comments - sometimes they can include information that will not
be shown, but is available from JS.
```

Autocorrection

When you read the wrong HTML, the browser automatically adjusts it for display and when building the DOM.

In particular, there will always be an upper <html> tag. Even if the text does not - in DOM it will, the browser will create it yourself.

The same goes for the <body> tag.

For example, if the file consists of one word "Hello", then the browser will automatically wrap it in <html> and <body>.

When generating the DOM, the browser itself handles errors in the document, closes the tags and so on.

Navigation on DOM-elements

DOM allows you to do anything with an HTML element and its contents, but you need to get the element you need first.

Access to the DOM begins with the document object. From it you can get to any knots.

The topmost tree elements are accessible directly from document.

```
<HTML> = document.documentElement
```

The first entry point is document.documentElement. This property refers to the DOM object for the <html> tag.

```
<BODY> = document.body
```

The second entry point is document.body, which corresponds to the <body> tag.

In modern browsers (except IE8-) there is also a document.head - a direct link to <head>

document.body can be null. You can not access an item that does not already exist when the script is executed.

In particular, if the script is in <head>, then document.body is not available in it.

Therefore, in the following example, the first alert displays null:

```
<!DOCTYPE HTML>
<html>
<head>
  <script>
    alert( "HEAD: " + document.body ); // null, body not yet
  </script>
</head>
<body>
  <script>
    alert( "BODY: " + document.body ); // body there is
  </script>
</body>
</html>
```

Here and further we will use two fundamentally different terms.

Child elements (or children) are the elements that lie directly within the given. For example, <HTML> usually contains <HEAD> and <BODY> inside.

Descendants - all the elements that lie within this, together with their children, the children of their children and so on. That is, the entire DOM subtree.

A pseudo-array (collection) childNodes stores all child elements, including text.

The example below will output the document.body children in series:

```
<!DOCTYPE HTML>
<html>
<body>
  <div>Stsrt</div>
  <ul>
    <li>Information</li>
  </ul>
  <div>End</div>
  <script>
    for (var i = 0; i < document.body.childNodes.length; i++) {
      alert( document.body.childNodes[i] ); // Text, DIV, Text, UL,
..., SCRIPT
    }
  </script>
  ...
</body>
</html>
```

Let's pay attention to a small detail. If you run the example above, the last element is <script>. In fact, there is still a text in the document (indicated by a ellipsis), but at the time the script was executed, the browser did not reach it yet.

All navigation properties can only be used for read-only reading. You can not just replace an element with an assignment

```
childNodes [i] = ....
```

The DOM is modified by other methods, which we will discuss later, all the navigation links are automatically updated.

The properties firstChild and lastChild provide quick access to the first and last element.

If there are child nodes, it is always true:

```
elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
```

DOM-collections, such as childNodes and others, which we will see below, are not JavaScript arrays.

They do not have methods for arrays, such as forEach, map, push, pop, and others.

That is why childNodes are called "collection" or "pseudo-mass".

To enumerate the collection, you should use the usual for loop:

```
for (var i = 0; i <elems.length; i ++)
```

Access to the elements on the left and right of this can be obtained from the links previousSibling / nextSibling.

The parent is accessible via parentNode. If you go from one element to another for a long time, sooner or later you will reach the root of the DOM, that is, before the document.documentElement, and then the document.

The navigation links described above apply equally to all nodes in the document. In particular, in childNodes both text nodes and nodes-elements and nodes-comments coexist, if any.

But for most tasks, text nodes are not interesting to us.

So let's look at an additional set of links that do not take them into account:

These links are similar to those that used to be, only in a number of places is the word Element:

- children - only child nodes-elements, that is, corresponding to the tags.
- firstElementChild, lastElementChild - respectively, the first and last children-elements.
- previousElementSibling, nextElementSibling - Neighboring elements.
- parentElement is the parent element.

Equalities are always true:

```
elem.firstElementChild === elem.children[0]
elem.lastElementChild === elem.children[elem.children.length - 1]
```

Special references for tables

Specific DOM elements can have their own additional links for greater ease of navigation.

Here we will consider the table, since this is an important special case and just an example.

The list below shows the most useful:

TABLE

- table.rows - collection of rows of the TR table.
- table.caption / tHead / tFoot - references to elements of the table CAPTION, THEAD, TFOOT.
- table.tBodies is a collection of table TBODY elements, there may be several of them according to their specification.

THEAD / TFOOT / TBODY

- tbody.rows is a collection of strings of the TR section.
- tr.cells - collection of TD / TH cells
- tr.sectionRowIndex - line number in the current section THEAD / TBODY
- tr.rowIndex - row number in the table
- td.cellIndex - the number of the cell in the line

Example of use:

```
<table>
  <tr>
    <td>one</td> <td>two</td>
  </tr>
  <tr>
    <td>three</td>  <td>four</td>
  </tr>
</table>

<script>
var table = document.body.children[0];

alert( table.rows[0].cells[0].innerHTML ) // "one"
</script>
```

Additional search methods

If an element is assigned a special id attribute, you can get it directly from the variable with the name from the id value.

For example:

```
<div id="content-holder">
  <div id="content">Element</div>
</div>


<script>
  alert( content ); // DOM-element
  alert( window['content-holder'] ); // in the name is hyphen,
                                     // so through [...]
</script>
```

This behavior corresponds to the standard. It exists, first of all, for compatibility, as a fragment of the distant past and is not very welcome, because it uses global variables. The browser tries to help us by mixing the JS and DOM namespaces, but conflicts are possible.

A more correct and common practice is to access the element by calling document.getElementById ("identifier").

For example:

```
<div id="content"> Select this item </div>
<script>
  var elem = document.getElementById('content');
  elem.style.background = 'red';
  alert( elem == content ); // true
  content.style.background = ""; // same element
</script>
```

By the standard, the id value must be unique, that is, there can be only one element in the document with this id. And it is he who will be returned.

If the document has several elements with a unique id, then the behavior is undefined. That is, there is no guarantee that the browser will return the first or last - it will return randomly.

Therefore, try to follow the rule of uniqueness id.

The elem.getElementsByTagName(tag) method searches for all elements with the specified tag tag inside the elem element and returns them as a list.

The tag register does not matter.

Unlike getElementById, which exists only in the document context, the getElementsByTagName method can search within any element.

For example, we find all the input elements inside the table:

```
<table id="age-table">
  <tr>
    <td>Your age:</td>

    <td>
      <label>
        <input type="radio" name="age" value="young" checked> under
the age of 18      </label>
      <label>
        <input type="radio" name="age" value="mature"> from 18 to
50
      </label>
      <label>
        <input type="radio" name="age" value="senior"> over 60
</label>
    </td>
  </tr>

</table>

<script>
  var tableElem = document.getElementById('age-table');
  var elements = tableElem.getElementsByTagName('input');

  for (var i = 0; i < elements.length; i++) {
    var input = elements[i];
    alert( input.value + ': ' + input.checked );
  }
</script>
```

You can get all the children by passing the asterisk '*' instead of the tag:

```
// get all the elements of the document
document.getElementsByTagName ('*');
// get all descendants of elem:
elem.getElementsByTagName ('*');
```

One of the most common mistakes beginners (however, sometimes not only) is to forget the letter "s", that is, try to call the getElementByTagName method instead of getElementsByTagName.

The letter "s" is not needed where there is only one element, that is, getElementById, in other methods it is required.

The call to document.getElementsByName (name) allows you to get all the elements with this name attribute.

For example, all elements named age:

```
var elems = document.getElementsByName('age');
```

Until the appearance of the HTML5 standard, this method returned only those elements that support the attribute name, in particular: iframe, a, input, and others. In modern browsers, the tag does not matter.

This method is used very rarely.

The call elem.getElementsByClassName (className) returns a collection of elements with class className. Finds an element also in the event that it has several classes, and the desired one is one of them.

Supports all modern browsers. For example:

```
div class="article">Article</div>
<div class="long article">Long article</div>

<script>
  var articles = document.getElementsByClassName('article');
  alert( articles.length ); // 2, find both elements
</script>
```

Like getElementsByTagName, this method can be called both in the context of the DOM element and in the context of the document.

Practical Work. Window Object (part 1)

Edit the HTML page (see below) as follows.

When you click the "Open Windows" button, ALL pages with tasks from module 1 should be open.

When you click the "Close Windows" button, all open pages should be closed.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style>
  .myclass {
    background-color: #CCFFFF;
    color: #CC0000;
    border: solid;
    border-color: #0000FF;
}
</style>
<title>John Smith</title>
<script>
```

```
// window object list
var arrWIN = [];
// URLs list
var arrURL = [];
arrURL[0] =
"http://cdo.kname.edu.ua/mod/assignment/view.php?id=129211";
arrURL[1] =
"http://cdo.kname.edu.ua/mod/assignment/view.php?id=129212";
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}

</script>
</head>
<body>
<script>
// functions
function openwin(){
  cnt.clr();
  for(var i=0; i<arrURL.length; i++){
    arrWIN[i] = window.open(arrURL[i]);
    cnt.add(i+": "+arrURL[i]+"<br>");
  }
}
function closewin(){
  for(var i=0; i<arrURL.length; i++){
    arrWIN[i].close();
  }
  cnt.out("No windows opened")
}
</script>
<form name="myForm">
<input name="open_win" type="button" onClick="openwin()"
value="Open windows">
<input name="close_win" type="button" onClick="closewin()"
value="Close Windows">
</form>
<hr />
Windows: <div class="myclass" id="my">No windows opened</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
</script>
</body>
</html>
```

Practical Work. Window Object (part 2)

Change the HTML-pages (see below) as follows.

1. For each number, the main window (mainwin.html) should open the child window (subwin.html), which will perform the factorization of the number into 2 prime factors.

2. The number is passed as a parameter to the URL, for example, subwin.html? P = 998027.

2. The child window writes the result to its HTML_Container object in the following form:

- number1x number2,
- number1 must be less than 2.
- x is a small Latin letter.

For example, the number 998027 is the product of numbers 991 and 997. In the answer it is necessary to write 998027 = 991x997

mainwin.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style>
  .myclass {
    background-color: #CCFFFF;
    color: #CC0000;
    border: solid;
    border-color: #0000FF;
}
</style>
<title> John Smith (main window)</title>
<script>
// window object list
var arrWIN = [];
// URLs list
var arrURL = [];
arrURL[0] = "subwin.html?p=129211";
arrURL[1] = "subwin.html?p=129212";
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}

</script>
</head>
<body>
<script>
// functions
function openwin(){
  cnt.clr();
  for(var i=0; i<arrURL.length; i++){
    arrWIN[i] = window.open(arrURL[i]);
    cnt.add(i+": "+arrURL[i]+"<br>");
  }
}
function closewin(){
  for(var i=0; i<arrURL.length; i++){
    arrWIN[i].close();
  }
  cnt.out("No windows opened")
}
</script>
<form name="myForm">
```

```
<input name="open_win" type="button" onClick="openwin()"
value="Open windows">
<input name="close_win" type="button" onClick="closewin()"
value="Close Windows">
</form>
<hr />
Windows: <div class="myclass" id="my">No windows opened</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
</script>
</body>
</html>
```

subwin.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<style>
  .myclass {
    background-color: #CCFFFF;
    color: #CC0000;
    border: solid;
    border-color: #0000FF;
}
</style>
<title> John Smith (child window)<</title>
<script>
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
</script>
</head>
<body>
Result: <div class="myclass" id="my">None</div>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
var param = location.search;
cnt.out(param);
</script>
</body>
</html>
```

Practical Work. Searching Items in the DOM

Part1

Change the HTML page index.html (see bellow) as follows.

Create a page index1.html (from index.html). When you click the button, you need to find all the "input" elements on the page and display the following information in the container:

The resulting collection (object).

For each item in the collection, print:

- collection item number,
- element itself (object),
- value of the element's value property,
- value of the element's name property,
- value of the cheked item property.

Part2

Change the HTML page index.html (see bellow) as follows.

Create a page index2.html (from index.html). When you click the button, you need to find all the elements with the names Res01 ... Res13 (names should be found in a loop, in the same loop search for a specific name) on the page and display all the elements of the collections received in the container.

For each collection, display the following information in the loop:

The resulting collection (object).

For each item in the collection, print:

- element number of the collection
- element (object) itself
- value of the element's value property
- the value of the element's name property
- the value of the cheked item property

Part3

Change the HTML page index.html (see bellow) as follows.

Create a page index3.html (from index.html). When you click the button, you need to find all the elements with classes class_01 ... class_03 (names should be found in a loop, in the same loop search for a specific name) on the page and display all the elements of the collections received in the container.

For each collection, display the following information in the loop:

The resulting collection (object).

For each item in the collection, print:

1. Item number of the collection
2. The very element (object).
3. List of children, for each child element
   - collection item number

- element itself (object)
- value of the element's value property
- value of the element's name property
- value of the cheked item property

index.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
<title>John Smith</title>
<style>
.class_01 {
  background-color: #FFFFE0;
}
.class_02 {
  background-color: #FF0000;
}
.class_03 {
  background-color: #CCCCCC;
}
</style>

<script>
// Constructor of output container object
function HTML_Container(elem){
  this.elem = elem;
  this.out = function(s){
    this.elem.innerHTML = s;
  }
  this.clr = function(){
    this.elem.innerHTML = "";
  }
  this.add = function(s){
    this.elem.innerHTML += s;
  }
}
</script>
</head>
<body>
<script>
  function process(){
    alert("Add necessary actions")
  }
</script>
<form name="myForm">
<input name="Process" type="button" onClick="process()"
value="Process">
</form>
<hr />
Result: <div class="myclass" id="my">conteiner</div>
<hr />
<form name="dialForm">
<table width="100%" border="1" cellspacing="1" cellpadding="5">
  <tr>
    <th width="20" scope="col" bgcolor="#FFFFE0">OK</th>
    <th width="20" scope="col" bgcolor="#FF0000">Err</th>
    <th width="20" scope="col" bgcolor="#CCCCCC">Not</th>
    <th width="50" scope="col">Pic</th>
    <th scope="col">Text</th>
  </tr>
<tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res01" value = "OK" /> </ td>
```

```
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res01" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res01" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im01.gif" /> </ td>
    <td valign = "top"> Multiple choice (closed question) with only
one correct answer (the student puts a mark in one of the circles)
    </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res02" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res02" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res02" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im01.gif" /> </ td>
    <td valign = "top"> Multiple choice (closed question) with one
or more correct answers (the student puts a mark in one or more
boxes)
  </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res03" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res03" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res03" value = "OK" checked /> </ td>
```

```
    <td align = "center" valign = "top"> <img src = "ttv_img /
im03.gif" /> </ td>
    <td valign = "top"> Alternative question (True / False) </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res04" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res04" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res04" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im04.gif" /> </ td>
    <td valign = "top"> Numeric question </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res05" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res05" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res05" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im05.gif" /> </ td>
    <td valign = "top"> Calculated question </ td>
  </ tr>
<tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res06" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res06" value = "OK" /> </ td>
```

```
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res06" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im06.gif" /> </ td>
    <td valign = "top"> Nested Answers </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res07" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res07" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res07" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im07.gif" /> </ td>
    <td valign = "top"> The question of correspondence </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res08" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res08" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res08" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im08.gif" /> </ td>
    <td valign = "top"> Short answer (open question) </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res09" value = "OK" /> </ td>
```

```
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res09" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res09" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im09.gif" /> </ td>
    <td valign = "top"> Description </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res10" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res10" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res10" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im10.gif" /> </ td>
    <td valign = "top"> Essays </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res11" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res11" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res11" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im11.gif" /> </ td>
    <td valign = "top"> The goal in the image </ td>
  </ tr>
  <tr>
```

```
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res12" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res12" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res12" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im12.gif" /> </ td>
    <td valign = "top"> All or Nothing </ td>
  </ tr>
  <tr>
    <td align = "center" valign = "top" class = "class_01"> <input
type = "radio" name = "Res13" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_02"> <input
type = "radio" name = "Res13" value = "OK" /> </ td>
    <td align = "center" valign = "top" class = "class_03"> <input
type = "radio" name = "Res13" value = "OK" checked /> </ td>
    <td align = "center" valign = "top"> <img src = "ttv_img /
im13.gif" /> </ td>
    <td valign = "top"> On the correspondence (displacement) </ td>
  </ tr>
</table>
</form>
<script>
// Globals
var elem = document.getElementById('my');
var cnt = new HTML_Container(elem);
</script>
</body>
</html>
```

References:

1. Бочаров Б.П. Інформаційні технології в освіті : монографія / Б.П. Бочаров, М.Ю. Воєводіна; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків: ХНУМГ ім. О. М. Бекетова, 2015. – 197 с.

2. Bocharov Boris. AUTOMATIZED WEB PAGES PARSING AND CREATION / Boris Bocharov, Maria Voevodina // Information technologies in education: electronic supplement to the journal "Educational Institutions Libraries". – 2017. – N5, p. 1-5.

3. Bocharov Boris. BASIC CONCEPTS OF THE JAVASCRIPT / Boris Bocharov, Maria Voevodina, Nataliia Braterska, Anastasiia Dashkovska // Information technologies in education: electronic supplement to the journal "Educational Institutions Libraries". – 2018. – N7, p. 1-48.