Introduction to JavaScript, Part 2

Luka Abrus
Technology Specialist, Microsoft Croatia

Interaction

In the first part of this guide, you learned how to use JavaScript, how to write code and how to see if it actually works. But that was all just basic stuff – in this second part we'll show you what JavaScript is really used for in the real world and where it can really make a difference.

Accessing page elements

As we've said before, JavaScript is primarily used for adding dynamic effects and user interactivity to your Web pages. One of the most important functions of JavaScript is to be able to access any element on the page. You can do this by creating a variable that represents that element. Then when you change the value of the variable, you are actually making changes to the element.

For example, if you want to check the text that a user has entered into a textbox, you need to access the textbox through code and programmatically check its contents. You can also hide and reveal elements on the page, or animate the text in the status bar using JavaScript.

Every HTML element can have an ID attribute. This is a unique identifier of that element and distinguishes it from all other elements on the page. Here's how the HTML tag for an element with an ID attribute would look:

```
<div id="myDiv">Hello!</div>
```

You shouldn't have two elements with the same ID on the same page - although the page would display properly, any script that would try to reference a single element with that ID wouldn't work correctly, as it wouldn't know which of the two elements to reference.

If you know the ID of an element, you can reference it easily. Here's how you could do it for the previous **DIV** element. Try copying this into an HTML file and opening this page in a web browser – what happens?

```
<script type="text/javascript" language="javascript">
<!--

alert(myDiv.innerText);

myElement = document.getElementById("myDiv");
alert(myElement.innerText);

//-->
</script>
```

In this example we have actually shown two ways to access elements. In the first case, we just used its ID ("myDiv") to display the text in the **DIV** element, and in the second case we created a new variable called "myElement" and given it the value of the myDiv element using the *document.getElementById* method. Although the first case seems much simpler, the second one is actually the preferred method because of its compatibility with all browsers and clarity as you won't make mistakes mixing your variables with element ID's.

We explained the *document* object in the first part of this guide and now we're using its *getDocumentById* method. This method returns a reference to the element with the ID provided in parenthesis. Watch out for the capitalization and remember that if an element with the provided ID name doesn't exist, the method will return a *null* value.

```
myElement = document.getElementById("n2309sd");
if (!myElement) {
    alert("This element doesn't exist!");
}
```

Note that if you try to execute this code before the element is defined on the page (before it is processed and displayed by the browser), such an element won't be found and the code will always return *null* value. So either put all your code at the end of the page after all HTML elements have been defined, or execute your code after the page has fully loaded (we'll show you how when we get to events in JavaScript).

Properties of HTML elements

When you have referenced an element and have a variable through which you can access it (like in our case, the variable *myElement*), you can easily access or change its properties. For example, with the *innerText* property you gain access to all the text that the element contains (our example would display "Hello!" as that is the content of the referenced **DIV** element).

The property of an element is written in lowercase and is separated by a dot (".") from the element name. Take the following element as an example.

```
<table id="myTable" width="300" height="500" border="0" cellpadding="5"
bgcolor="red"><tr><td>Hello!</td></tr></table>
```

At the moment it isn't important which properties have been defined for this element, but rather how you work with them in code. If you follow the previous instructions, you could access the *border* property by adding ".border" to element name in your JavaScript code. And the same goes for all other properties:

```
table = document.getElementById("myTable");


TableWidth = table.width;
TableHeight = table.height;
TableBorder = table.border;
TableCellPadding = table.cellPadding;
TableBgColor = table.bgColor;
```

In this example, we have created multiple different variables and given them the values of the table element properties. One quick note about property name standards – if a certain HTML element property name consists of two or more words strung together (like "cellpadding"), then when you access that element in JavaScript, the first word is written with first letter in lowercase, while other words have first letter in uppercase (like "cellPadding", "bgColor", and even the "getElementById" method).

If you want to change the value of an HTML element property, simply assign it a new value in your JavaScript code:

```
table.border = "4";

table.bgColor = "blue";
```

Try copying and pasting the HTML table element above into an HTML page and view it in a browser. Now copy the JavaScript code to dynamically change the HTML property values through the JavaScript *table* variable. View this page again in a browser – did the table properties change?

You'll also often use the *style* property to define other properties of an element, like in the following example:

```
<table id="myTable" style="width: 300px; height: 500px; border: solid 1px black;
background-color: Red;"> . . . </table>
```

To access these properties, you just have to reference them through ".style" object which contains all of these properties.

```
table = document.getElementById("myTable");

TableWidth = table.style.width;
```

All of these properties are read-write (readable and writeable), so you can dynamically change their values in the code, and the changes will be instantly visible on the page.

Hiding and displaying elements

As you've seen, you can easily modify the style properties of an element. If you're familiar with CSS (if you're not, we suggest you read our "Introduction to CSS" guide), you can now make your Web page much more dynamic by changing the CSS properties.

In this part, we'll focus on the *display* CSS property that is used for displaying or hiding an element. We'll use a **DIV** element, which we can display and hide using JavaScript code. First, let's take a look at the **DIV** element:

```
<div id="myDiv" style="display: none;">

This element isn't displayed on the page!

</div>
```

We've set its *display* property to *none*, and thus made it invisible so it won't be displayed on the page. You can apply this CSS property to any HTML element, whether it is an image, a table or anything else.

We'll dynamically change its *display* properties, which will make it visible. First we need two methods that will display it and hide it:

```
function hide(elementID) {

    var myElement = document.getElementById(elementID);

    if (myElement) myElement.style.display = "none";
}


function show(elementID) {
    var myElement = document.getElementById(elementID);
    if (myElement) myElement.style.display = "block";
}
```

These are generic methods that take the ID of an element as a parameter and then, if an element with such an ID exists (`if (myElement)`), the method will display it or hide it by changing its *display* CSS property (`myElement.style.display = "block"`).

All we have to do now is to call these methods from the page. We'll create two links that execute JavaScript for showing and displaying an element, but as you'll see later in this guide, you can create buttons or anything else and then attach an event to it. For now, we'll keep it simple – here are the links:

```
<a href="javascript:hide('myDiv');">Hide it!</a>
<a href="javascript:show('myDiv');">Show it!</a>
```

By clicking any of the links, we call appropriate method with *"myDiv"* as a parameter (the ID of the element we want to hide or display).

Quiz 1

Question 1.1: You have defined the following HTML element:

```
<span id="text1">My element</span>
```

How can you access it through code and what do you have to be careful about?

Answer 1.1: You can access it both by directly referencing its name, and by using the *document.getElementById* method. For example, if you want to change its contents:

```
text1.innerText = "This is my element";


t1 = document.getElementById("text1");
t1.innerText = "This is my element";
```

You have to be very careful about the variable names. If you already have a variable in your code with the same name as the ID of an HTML element, your code will result in an error. For example, the following code wouldn't work on the same page, because *text1* is used as an HTML element ID as well as a JavaScript variable:

```
text1 = "Hello";
text2 = "world"
text = text1 + " " + text2;
```

Question 1.2: You have the following HTML element with CSS properties defined on your page:

```
<div id="myDiv" style="width: 120px; border: solid 1px black;">My element</div>
```

How can you change its width (CSS property) to 400 pixels?

Answer 1.2: Just access it the usual way, but be careful – you have to specify the full property value. In CSS that is both the value and the unit, so you have to write "400px":

```
myDiv.style.width = "400px";
```

Question 1.3: Based on the example with hiding and showing elements, write a script that would also hide and display the link for hiding / displaying the element. Make sure that at any time only one appropriate link is visible (for example, if the element is hidden, display only "Show it!" link and vice versa).

Answer 1.3: Our first step is, of course, to create the element which will be displayed and hidden:

```
<div id="myDiv" style="display: none;">Hello!</div>
```

Next we have to add few more properties to our links. We need to give them ID's to be able to reference them from code. We also need to hide the "Hide it!" link, as the element will be initially hidden when the page loads.

```
<a href="javascript:hide('myDiv');" id="hideLink" style="display: none;">Hide
it!</a>

<a href="javascript:show('myDiv');" id="showLink">Show it!</a>
```

The last thing is to upgrade our methods for hiding and displaying the element. We just need to add code for hiding and displaying appropriate links (if we're hiding the element, we need to display the "Show it!" link and hide the "Hide it!" link).

```
function hide(elementID) {
    var myElement = document.getElementById(elementID);
    if (myElement) myElement.style.display = "none";
    showLink.style.display = "block";
    hideLink.style.display = "none";
}

function show(elementID) {
    var myElement = document.getElementById(elementID);
    if (myElement) myElement.style.display = "block";
    showLink.style.display = "none";
    hideLink.style.display = "block";
}
```

Events

JavaScript is an event-driven scripting language. That means you can interact with all the events in the browser and react to them. Examples of some events are when user moves his mouse over an element, when he clicks a button, even when the page loads or closes. Being able to react to events is crucial for interaction between the page and the user.

In the following table you can find the most important events in JavaScript with short descriptions. Note that all events which affect the *document* element can also be applied to any element on the page (i.e. the **B** element, **TD** element, **IMG** element, etc.).

Event	**Elements affected**	**What starts this event?**
onabort	Image	interruption of image loading (user has clicked on another link or STOP in the browser)
onblur	Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window	when element loses focus (another element is selected)
onchange	FileUpload, Select, Text, Textarea	change of data inside an element
onclick	Button, Document, Checkbox, Link, Radio, Reset, Submit	click on an element
ondblclick	Area, Document, Link	double click on an element
ondragdrop	Frame, Window	drag and drop of a shortcut or a file to the browser window
onerror	Frame, Image, Window	error in the script or while loading an image (e.g. image not found)
onfocus	Button, Checkbox, FileUpload, Frame, Layer, Password, Radio, Reset, Select, Submit, Text, Textarea, Window	when an element gets focus (opposite of onblur)
onkeydown	Document, Image, Link, Textarea	pressing down a key on the keyboard
onkeypress	Document, Image, Link, Textarea	pressing (and releasing) a key on the keyboard
onkeyup	Document, Image, Link, Textarea	releasing a key on the keyboard
onload	Frame, Image, Layer, Window	end of loading
onmousedown	Button, Document, Link	pressing mouse button
onmouseout	Area, Layer, Link	mouse pointer exiting the element’s area
onmouseover	Area, Layer, Link	mouse pointer moving over an

		element
onmouseup	Button, Document, Link	releasing mouse button
onmove	Frame, Window	window or frame moving
onreset	Form	resetting the form
onresize	Frame, Window	changing size of a window or a frame
onselect	Textarea	selecting text
onsubmit	Form	submitting a form
onunload	Frame, Window	unloading a document (leaving the page or closing the window)

Roughly speaking, all events can be split into three groups: events that react to mouse action (like *onMouseOver* or *onMouseDown*), those that react to the keyboard (like *onKeyDown* or *onKeyPress*) and those that are started by events on the page itself (like sending the form or resizing the window).

You might have noticed that events aren't unique. For example, pressing a keyboard button will start *onKeyDown* and *onKeyUp* events (both of which are incorporated by *onKeyPress* event).

If you want to attach an event handler to an HTML element and define the action that will start when the event occurs, you'll need to add the event as a property of the element.

```
<b onclick="alert('You\'ve just clicked this bold text!')">Bold text.</b>
```

Inside of its *onclick* property (note that this property name matches the event name you want your code to respond to) we can just enter the JavaScript code to be executed in quotes. Just remember to watch out for those single and double quotes!

Our example is fairly simple – when the user clicks this element on the page, he is presented with a message box. Copy and paste this into an HTML page and try it yourself. Consult the events table listed above to attach events and code to your elements.

You can also call specify more complex JavaScript commands within the event handler. Just separate each command with a semi-colon.

```
<b onclick="myFunctionCall();alert('Hello!');">Bold text.</b>
```

The event you're most likely to use is *onclick*, but we would like to point one more very useful event. The *onload* event is started when an element finishes loading. This is applicable to the web page as a whole, because you'll want to know when your page has completely loaded so you can start executing your JavaScript code.

```
<body onload="startSomething()">
```

Just apply it to the **BODY** element and attach a call to the main method of your code. You can now put all your JavaScript methods into this main method (in this case the *startSomething* method) and start executing right after the page has finished loading.

Quiz 2

Question 2.1: How would you display a message to the user when he's leaving the page? For example, when he's moving to another page or closing the browser?

Answer 2.1: Use the *onunload* event with the **BODY** element:

```
<body onunload="alert('Goodbye!');">
```

Question 2.2: You have two of the same images, one in color and the other in black and white. You want to initially display the black and white image on the page, but when the user's mouse moves over it, the image should be swapped with the colored one. As the mouse leaves the image area, the images should change back to the black and white version.

Answer 2.2: We will use the *onmouseover* and *onmouseout* events. In JavaScript we can use *this* object to reference the current object (which in this case is the IMG element). Look at the solution:

```
<img src="img_bw.gif" width="290" height="30"
onmouseover="this.src='img_color.gif'" onmouseout="this.src='img_bw.gif'" />
```

We just changed the *src* property of the current image (referenced with *this* object). We could have also done it the longer way:

- add an ID to the image element
- create two methods that would be called with the *onmouseover* and *onmouseout* events
- and then change the *src* property of the IMG element (referenced by its ID name) in these methods.

Question 2.3: Let's say you want to display an image from another server, but you want to be prepared for the worst – if the image cannot be loaded (for example, the server it is hosted on is not working or the image has been deleted), you want to display your own backup image from your server. How would you do it?

Answer 2.3: Use the *onerror* event – if the image loading results with an error, you can change its *src* property to the local image and display it.

```
<img src="http://error_server.com/non_existent_image.gif" width="290" height="30"
onerror="this.src='my_image.gif'" />
```

Working with forms

Forms are a crucial part of many Web pages. You can use them to collect orders from customers, to register users or, simply put, to collect any kind of information from your visitors. JavaScript can, of course, access all the form contents and do something with the data. Most often you'll check and validate what the user has entered and point him to the missed form fields or inappropriate data.

To access a form on the page you can use the *document.formName* syntax. For example, if you have the following form on your page:

```
<form name="myForm" method="post" action="myScript.aspx">
...
</form>
```

The key is the form name, in our case, *myForm*. You can access this form with the *document.myForm* syntax. Forms in JavaScript are objects that model the structure of the form in HTML. To show you this, we'll create a form validation script and check the data entered into the form before it is sent to the server.

In this guide we won't discuss the server-side scripts that accept the data and work with it (for example, these server-side scripts could write the data into a database or send it in e-mails). Such scripts could also do some data validation, but it's encouraged to do most of it in JavaScript for performance reasons. In this example, all the validation is done with JavaScript in the browser, so there is no need to send the data to the server, wait for it to be processed, and then return with an error message to the user.

Bear in mind that JavaScript validation shouldn't be the only validation you do, as some users might turn JavaScript off in their browsers. JavaScript is used here just for the responsiveness of the Web page and for a better user experience.

Let's take a simple form as an example. We'll request the following information from the user – name, e-mail address, age, and job title. We want to make sure all this information is correctly entered into the form. The first three fields (name, e-mail address and age) will be regular text fields, while the job title will be a dropdown menu.

Here's how the HTML code for such a form would look:

```
<form name="myForm" method="post" action="myScript.aspx">
    <input name="name" type="text" size="30">
    <input name="email" type="text" size="30">
    <input name="age" type="text" size="3">
    <select name="job">
       <option>--- select one ---
       <option value="student">Student</option>
       <option value="businessperson">Business Person</option>
       ...
    </select>
    <input type="submit" value="Send!" onClick="checkForm(); return false;">
</form>
```

This is a simplified version of a form in HTML – in a full version you would definitely put text in front of each form field to specify what kind of data you expect to be entered there, and you would probably give a longer list of possible jobs (that's what those three dots stand for).

Try copying this form into an HTML page and adding description text to each field, line breaks and more job type options in the dropdown menu.

The key part of this form is the submit button. We've attached an event handler that handles a mouse click on the submit button, and in this case it would also handle a key press event which occurs if the user navigates to it with a keyboard (by pressing tab key) and then presses Space or Enter key.

Basically, whenever the user wants to submit this form, we call our *checkForm* method that does the validation. We are also canceling the event – by using the "*return false"* statement in our *onClick* script, we cancel any further execution of submit event. This way we have stripped the submit button of its basic functionality (submitting the form to the server-side script) and added the call to our method. Don't worry, we will submit the form through our JavaScript code, but only if all the data is entered correctly.

So what should our checkForm method check? What are the basic requirements on this form? In the field *name* the user should enter at least two words separated with a space (first name and last name), so we will check for at least one space character in the value. The e-mail address should contain the characters "@" and "." (again, this is a minimal check – doesn't ensure that this is actually a valid email).

The contents of the *age* field should be a number greater than zero. From the jobs dropdown menu, the user can select any option, but not the first one, as it serves as instructions and has no other purpose.

Here's how our *checkForm* method should look:

```
function checkForm() {
    var formName = document.myForm.name;
    var formEmail = document.myForm.email;
    var formAge = document.myForm.age;
    var formJob = document.myForm.job;

    if (formName.value.indexOf(" ") == -1) {
        alert("Please enter your full name.");
        formName.focus();
    } else if (formEmail.value.indexOf("@") == -1 || formEmail.value.indexOf(".")
== -1) {
        alert("Please enter a valid e-mail address.");
        formEmail.focus();
    } else if (! (parseInt(formAge.value) > 0)) {
        alert("Please enter your age as a valid nuber.");
        formAge.focus();
    } else if (formJob.selectedIndex == 0) {
        alert("Please select your job from the list.");
        formJob.focus();
    } else {
        document.myForm.submit();
    }
}
```

Let's explain this script part by part, as it introduces a few new methods and programming concepts. The first part is rather simple – we load the variables *formName*, *formEmail, formAge* and *formJob* with references to the appropriate HTML form fields. As we said before, forms are actually just objects in JavaScript, and to reference their fields, we just write the field name after the form name, for example:

```
var formName = document.myForm.name;
```

To access the value of the form field, we use its *value* property. So to get the current contents of, for example, the *email* field, we could use any of the following two approaches:

```
text = document.myForm.email.value;


var formEmail = document.myForm.email;

text = formEmail.value;
```

As we've already created JavaScript variables with references to the HTML form fields, we'll use the second approach. Now we have to check the contents of the fields. We use the *indexOf* method to check if a specific character exists within the form field content. If we realize the field doesn't contain valid data, we display an appropriate message and place the focus on that field. To do this, we use the *focus* method, which places the text cursor in that field, making it easy for the user to just start typing in the correct value.

To convert a text value to a number we use the *parseInt* method. It's a built-in JavaScript method that takes text as a parameter and returns a valid whole number if possible, or a *NaN* value (Not A Number) if the conversion is not possible. For example, if you try converting "abc", you will get a *NaN* value as a result. But if you try to convert "15", "-58", you will get 15 or -58 number values respectively.

Checking the *job* field is a bit different. As it doesn't contain text, but rather a dropdown menu, we need to find out which option is selected. We do it with the *selectedIndex* property, which returns the index of the selected option. These indexes are zero-based, which means the first option has an index of 0, second one has index 1, third one has index 2, etc. So if the selected option is the first one (with the text "--- select one ---"), the *selectedIndex* will have the value of 0. Again, if that is the case, we display an error message and focus on the dropdown menu.

At the end, if no *if* statements are met (which means that all the data has been correctly entered), the final *else* statement is executed. There we call *submit* method on the form, which, of course, submits the form:

```
document.myForm.submit();
```

To repeat once again – we have put the *return false* call in the *onClick* event handler of the submit button, which stops the form from automatically submitting. That's why we have to manually submit the form, which is done by calling the *submit* method on the form.

Try copying the checkForm function into the JavaScript section of your HTML page and viewing it in a browser. Now try entering in incorrect data – do you get the appropriate error messages? Note that submitting the form doesn't do anything since we haven't defined where this data should be received.

When creating a form, make sure you specify which fields are necessary (use bold or colored text, or put a "*" sign by each field). If you want the user to follow a certain format (for example, to write the date in mm/dd/yyyy format, or the phone number in "+x (xxx) xxx-xxxx" format), just make sure you specify that by the field. This way the user will know from the start what kind of data he/she needs to enter.

Quiz 3

Question 3.1: How would you create a standard button (with type set to "button", not to "submit") that would serve as a submit button after doing the data validation?

Answer 3.1: Since we actually stripped the submit button of its functionality, we can just create a standard button and in its *onclick* event handler call the *checkForm* method. Our method does the form submission by calling *document.myForm.submit* method, so we're all set. Just replace the submit button with the following code:

```
<input type="button" value="Send!" onClick="checkForm();">
```

Question 3.2: In our form validation example, write the *if* clause that would limit the age the user enters to a value between 8 and 88.

Answer 3.2: To make our solution a bit simpler, we'll first convert the value of the *age* field with the *parseInt* method and then compare it with the desired values:

```
formAgeValue = parseInt(document.myForm.age.value);
...
//note that || denotes a logical OR
else if (formAgeValue < 8 || formAgeValue > 88) {
    alert("Age must be between 8 and 88!");
    formAge.focus();
} ...
```

Question 3.3: Add new functionality to the *name* field. If the user enters "John Doe" (use the *onchange* event handler), fill the rest of the form with specific values - set age to 21, e-mail address to "john@doe.com" and select the second dropdown option ("Student").

Answer 3.3: First we need to change the *name* field and add the *onchange* event handler so that any change of the value calls our method.

```
<input name="name" type="text" size="30" onchange="checkName();">
```

If the user enters something into this field (and thus changes its value which triggers the *onchange* event), the *checkName* method will be called. In it, we just have to check the current value of the *name* field and if it is equal to "John Doe", fill in the other fields.

```
function checkName() {
    if (document.myForm.name.value == "John Doe") {
        document.myForm.email.value = "john@doe.com";
        document.myForm.age.value = "21";
        document.myForm.job.selectedIndex = 1;
    }
}
```

Again, selecting a dropdown menu option is done through the *selectedIndex* property. And you need to set it to the appropriate index (as "Student" is the second option, it needs to have the index set to 1).

Working with windows

JavaScript can be used to open new browser windows and to configure their properties. We'll show you how to do that and what can be done in just one line of code. Let's introduce the *window* object first, and its *open* method. We'll talk more about the *window* object later on in this guide, but for now let's move to the main thing – opening new windows with the *open* method.

The *open* method takes three parameters. The first parameter is the address of the page that will be displayed in the newly opened window, the second parameter is the name of the window, and the third parameter describes the properties of the new window. The result of the *open* method is a reference to the newly opened window. With this reference we can make additional adjustments to the window properties, like resizing and repositioning the window on the screen, as you'll see later.

Take a look at the example:

```
myWindow = window.open("myPage.html", "myPage", "height=300,width=400");
```

This way we opened the page "myPage.html" in a new window that has the height of 300 pixels and width of 400 pixels.

Let's look at a similar method of opening a link in a new window by using *target="_blank"* property of the link:

```
<a href="myPage.html" target="_blank">Open window!</a>
```

This would also open the "myPage.html" in a new window, but this way you cannot control the properties of the new window (for example, the width and height like in the previous example), nor can you later access this window through JavaScript.

Once you have opened the new window, you can easily load new Web pages in it with standard links. Just set their *target* property to the name of the newly opened window (the second parameter of the *window.open* method). So if you have already opened a new window with the name "myPage", this link would open the page "mySecondPage.html" in that window:

```
<a href="mySecondPage.html" target="myPage">My link</a>
```

Bear in mind that if you haven't previously opened a window named "myPage" with the *window.open* method, this link would open a new standard window called "myPage" without any special properties. So, if you do want to specify certain properties of your new windows, just make sure that you actually create them through JavaScript before you load any Web pages into them.

So what properties of browser windows can you set? Take a look at the following table with the often used properties:

Property	**Description**
fullscreen	Opens the window in full-screen mode
height	Height of the new window in pixels. Minimum is 100.
location	Displays the address bar.
menubar	Displays the menu bar (File, Edit…).
resizable	Enables new window to be resizable.
scrollbars	Sets which scrollbars (vertical, horizontal) will be displayed, if necessary.
status	Display the status bar.
toolbar	Display the browser toolbar.
width	Width of the new window in pixels. Minimum is 100.

If you don't specify a property, it won't be used. So, for example, if you don't specify the *location* property, the address bar won't be displayed. Take a look at the following example – it opens up a new window with only the menu bar displayed.

```
myWindow = window.open("myPage.html", "myPage", "menubar=1");
```

Properties take different values – they can take Boolean values ("0" / "1" or "no" / "yes") or integer values (for width and height). So if you don't want a specific property turned on, you can either leave it out, or specify it with the value "0" or "no". Also, as the previous table states, the *width* and *height* properties cannot take a value that is less than 100 (such a window would be too small).

All properties are separated with a comma, and make sure there is no space between the properties listed in the third parameter. Here's how you would open a window 300 pixels wide with a status bar and a toolbar:

```
myWindow = window.open("myPage.html", "myPage", "width=300,status=1,toolbar=1");
```

Try copying the JavaScript above into an HTML page (in the JavaScript section), and instead of opening myPage.html, insert the URL of your favorite website. View this page in a browser – what happens? What does the new browser window look like? Now try modifying the window properties.

In each example we've created the reference to the new window in the variable *myWindow*. With this reference, we can do modifications to the newly opened window. For example, if we want to close it, we just need to call its *close* method:

```
myWindow.close();
```

There are two more useful methods that will give you full control over the opened window. If you want to move the window, use the *moveTo(x,y)* method that will move the window to the location set by the two parameters. If you want to move the new window to the top left corner (the coordinates of the top left corner are 0,0), just write:

```
myWindow.moveTo(0,0);
```

You can also resize the window – to do this, use the *resizeTo(width, height)* method. For example, if you want to resize the window to 200 x 200 pixels, write:

```
myWindow.resizeTo(200,200);
```

You can even apply these methods to the current window. For that, you need to use the *window* object itself. So if you want to resize the current window and move it to the top left corner, here's how to do it:

```
window.resizeTo(1000,1000);

window.moveTo(0,0);
```

Now, we'll show you a little trick. There's another useful object when working with windows – the *screen* object. It has 2 interesting properties, *availWidth* and *availHeight* which return the full available width and height on the screen. So if you want to resize the current window to the full available width and height of the screen, you can call the following method:

```
function maxWindow() {

    window.moveTo(0,0);

    window.resizeTo(screen.availWidth, screen.availHeight);

}
```

The available width and height of the screen depend on the different desktop toolbars or taskbars that the user has turned on (for example, the Sidebar in Windows Vista, or the Windows Taskbar, etc.).

Quiz 4

Question 4.1: Open a new window that displays the status bar, set its width and height to 500 pixels and make it not resizable by the user.

Answer 4.1: Use the following code:

```
myWindow = window.open("myPage.html", "myPage", "status=1,width=500,height=500");
```

As we don't want to make the window resizable, it's enough to omit this property.

Question 4.2: How would you set the *target* property of the links to make them open in the previously (question 4.1) opened window? Would you use "myWindow" or "myPage" for the target?

Answer 4.2: You need to use the name of the newly opened window as the target. That is the second parameter of *window.open* method, so you need to use "myPage".

```
<a href="WebPage.html" target="myPage">My link</a>
```

Question 4.3: Write the method that resizes the window opened in question 4.1 to 300x300 pixels. Ask the user if he prefers the new size and if he answers "No", resize the window back to its original size.

Answer 4.3: The key to this task is presenting the question to the user in the appropriate window. If you present the "Yes/No" option (by using *confirm* method) in the original window, the user won't be able to see the size of the newly opened window. So we have to display the *confirm* method in the resized window. Here's how:

```
function resizeWindow() {
    myWindow.resizeTo(300,300);
   //note that the "!" symbol indicates negation (i.e. the user clicked No)
    if (!myWindow.confirm("Do you want to keep new window size?")) {
        myWindow.resizeTo(500,500);
    }
}
```

We just call the *confirm* method that is available in the *myWindow* object (which serves as a reference to the new window). Try it out!

Conclusion

As you've seen, JavaScript is a very powerful and flexible scripting language that adds interactivity to the browser and enhances the user experience. Consider this guide as only the beginning of your JavaScript adventures.

JavaScript offers a lot more and we encourage you to learn by yourself, experimenting with and modifying the scripts shown in this guide. Discover more JavaScript possibilities in the MSDN documentation and tutorials available here - http://msdn.microsoft.com/workshop/author/dhtml/reference/dhtml_reference_entry.asp. Study different properties of HTML elements available through JavaScript and look through the tutorials to get fresh ideas for your Web pages. Happy scripting!

Author bio: Luka Abrus works for Microsoft Croatia as Technology Specialist. Prior to coming to Microsoft, Luka has worked as Web developer with several companies and as an author for Croatian IT magazines writing over hundred articles. He has written three books on Web development and Microsoft .NET. In his spare time, Luka is active in sports and playing piano.