JavaScript

Client Side

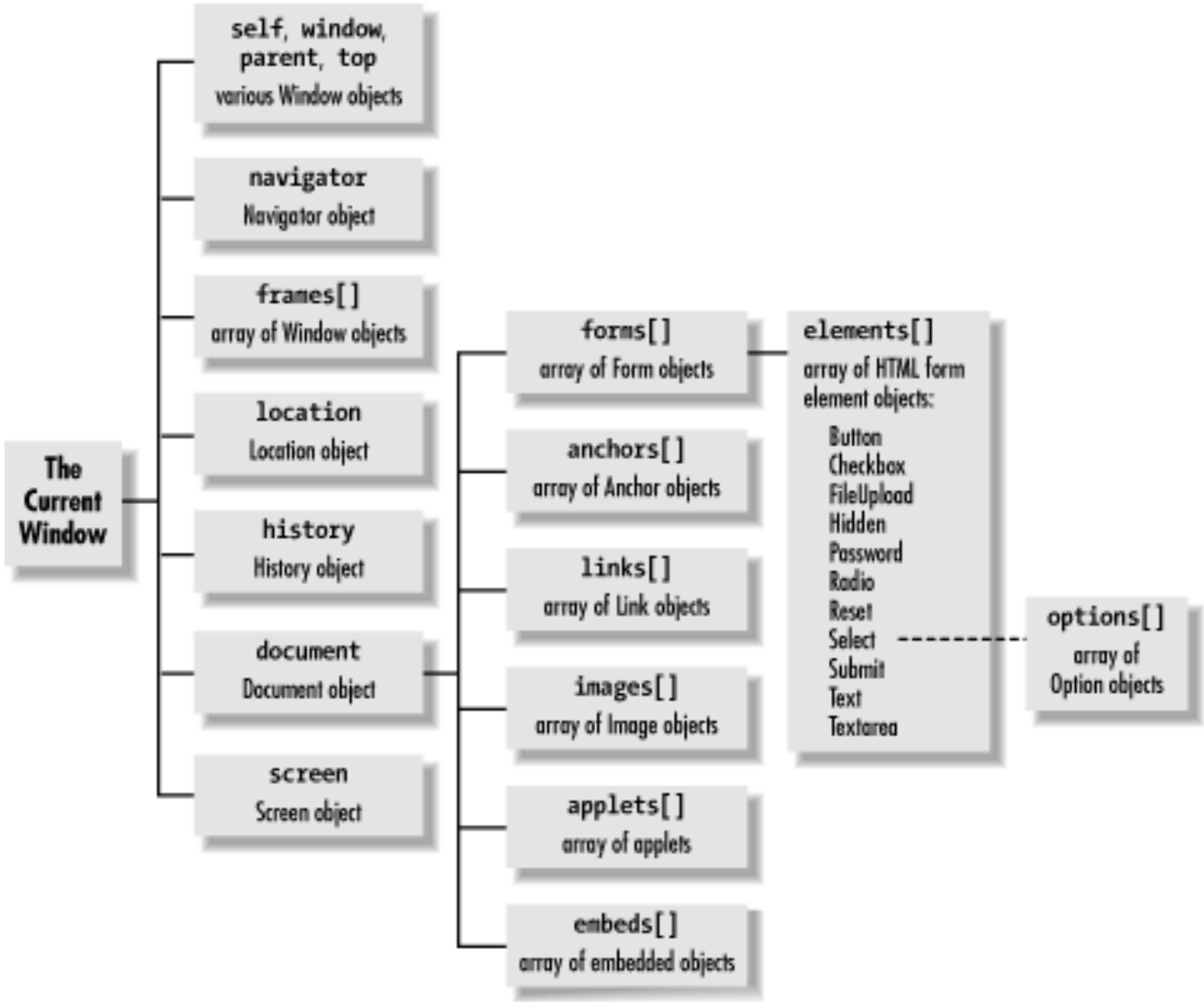
Francesco Marcelloni

and Alessio Vecchio

Dipartimento di Ingegneria dell'Informazione
Università di Pisa
ITALY

Client-side objects

When you load a document in a Browser, it creates a number of JavaScript Objects.



Client-side objects
Objects in a Page

Every page has the following objects

- navigator: has properties for the name and version of the user agent being used, for the MIME types supported by the client, and other info about the execution environment of the client.
- window: the top-level object; has properties that apply to the entire window. There is also a *window* object for each "child window" in a frame document.
- document: contains properties based on the content of the document, such as title, background color, links and forms.
- location: has properties based on the current URL.
- history: contains properties representing URLs the client has previously requested.

Client-side objects
Controlling objects

By controlling client-side objects, we can

- Open new browser windows, and determine their size and whether they should have scroll bars, or location windows
- Change the content of frames, even rewriting the HTML in a frame without downloading a new file
- Add or remove text from forms including text areas and select boxes
- Control the background color of your Web pages using JavaScript
- Obtain information about the execution environment
  - geolocation
  - number of cpus, memory

Client-side objects
Window Object

- Every browser window that is currently open will have a corresponding **window** object.
- If a document contain frames (<iframe> tags), the browser creates one window object for the HTML document, and one additional window object for each frame.
- All the other objects are children of one of the window objects, except for **navigator** and **screen**.
- In browsers that support tabs, every tab has its own window object.
- Some properties of window are not related to the user interface window. The window object provides global access to them.

Client-side objects
Window Object

- Every window has a history of the previous pages that have been displayed in that window, which are represented by the various properties of the **history** object.
- JavaScript has an implicit idea of the current window:
  - almost all references to sub-objects of the current window do not need to refer the window object explicitly.
  - This is why all of our output has been done using *document.write()* rather than *window.document.write()*.

Client-side objects
Window Object Properties

Property	Description
closed	Returns a Boolean value indicating whether a window has been closed or not
~~defaultStatus~~	~~Sets or returns the default text in the statusbar of a window~~
document	Returns the Document object for the window
frames	Returns an array of all the frames (including iframes) in the current window
history	Returns the History object for the window
innerHeight	Sets or returns the inner height of a window's content area
innerWidth	Sets or returns the inner width of a window's content area
length	Returns the number of frames (including iframes) in a window

Client-side objects
Window Object Properties

Property	Description
location	Returns the Location object for the window
name	Sets or returns the name of a window
navigator	Returns the Navigator object for the window
opener	Returns a reference to the window that created the window
outerHeight	Returns the outer height of a window, including toolbars/scrollbars
outerWidth	Returns the outer width of a window, including toolbars/scrollbars
pageXOffset	Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
pageYOffset	Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window

Client-side objects
Window Object Properties

Property	Description
parent	Returns the parent window of the current window
screen	Returns the Screen object for the window
screenLeft	Returns the x coordinate of the upper-left hand corner of the window (Explorer – Opera), same as screenX
screenTop	Returns the y coordinate of the upper-left hand corner of the window (Explorer – Opera), same as screenY
screenX	Returns the x coordinate of the upper-left hand corner of the window (Firefox – Chrome), same as screenLeft
screenY	Returns the y coordinate of the upper-left hand corner of the window (Firefox – Chrome), same as screenTop
self	Returns the current window, useful in web workers
~~status~~	~~Sets the text in the statusbar at the bottom of a window~~
top	Returns the topmost browser window

Client-side objects
Window Object Properties

Method	Description
alert()	Displays an alert box with a message and an OK button
blur()	Removes focus from the current window
clearInterval()	Clears a timer set with setInterval()
clearTimeout()	Clears a timer set with setTimeout()
close()	Closes the current window
confirm()	Displays a dialog box with a message and an OK and a Cancel button
focus()	Sets focus to the current window

Client-side objects
Window Object Methods

Method	Description
moveBy()	Moves a window relative to its current position
moveTo()	Moves a window to the specified position
open()	Opens a new browser window
print()	Prints the content of the current window
prompt()	Displays a dialog box that prompts the visitor for input
resizeBy()	Resizes the window by the specified pixels
resizeTo()	Resizes the window to the specified width and height

Client-side objects
Window Object Methods

Method	Description
scrollBy()	Scrolls the content by the specified number of pixels
scrollTo()	Scrolls the content to the specified coordinates
setInterval()	Calls a function or evaluates an expression at specified intervals (in milliseconds)
setTimeout()	Calls a function or evaluates an expression after a specified number of milliseconds

Client-side objects
Window Object Methods

```
<head>
<meta charset="utf-8">
<title> Object Model Example </title>
<script>
let myWindow;
function openWin() {
  myWindow=window.open('','','width=200,height=100');
  myWindow.document.write("This is 'myWindow'");
}
function closeWin() {
  myWindow.close();
}
function moveWin() {
  myWindow.moveBy(250,250);
  myWindow.focus();
}
</script>
</head>
```

Client-side objects
Window Object Methods

```
<body>
<div>
<input type="button" value="Open 'myWindow'" onclick="openWin()" >
<br><br>
<input type="button" value="Move 'myWindow'" onclick="moveWin()" >
<br><br>
<input type="button" value="Close 'myWindow'" onclick="closeWin()" >
</div>
</body>
```

open() arguments:

- **url**: the url of the resource to be loaded, if '' a blank page is shown
- **target**: the name of the browsing context, if it does not correspond to a known one, a new context is created; _self, _blank, _parent, and _top can be used.
- **windowFeatures**: a string that defines the properties of the new window as a set of name=value pairs. If windowFeatures are provided in general a popup window is created.

Client-side objects
Navigator Object

- The navigator object contains information about the browser.
- The JavaScript runtime engine on the client automatically creates the navigator object.
- Use the navigator object to determine which version of the browser your users have, what MIME types the user's Navigator can handle, its position, device memory, number of cpus, etc
- All of the properties of the navigator object are read-only.

Client-side objects
Navigator Object properties

- **userAgent**: returns the user agent string, identifies the browser; can be configured by the user via the browser's settings; so it is not reliable
- **appCodeName** and **appName**: are always "Mozilla" and "Netscape", deprecated.
- **geolocation**: gives access to the position of the device
- **hardwareConcurrency**: a number between 1 and the number of logical processors
- **cookieEnabled**: returns a boolean if cookies are enabled or not

Client-side objects
Navigator Object (example 1)

```
<script>
document.write("Code Name: " + navigator.appCodeName + "<br>") ;
document.write("Name: " + navigator.appName + "<br>" );
document.write("User Agent: " + navigator.userAgent + "<br>") ;
</script>
```

Output:

Code Name: Mozilla

Name: Netscape

User Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36

Client-side objects
Navigator Object (example 2)

```
let infobrowser = "<h1>Information on your browser</h1><p>";
for (let proprietyName in navigator)
  infobrowser += "<strong>" + proprietyName + ":</strong>" +
  navigator[proprietyName] + "<br>";
infobrowser += "</p>";
document.writeln(infobrowser);
```

Client-side objects
Navigator Object (example 2)

Information on your browser
vendor: Google Inc.
maxTouchPoints:0
scheduling:[object Scheduling]
userActivation:[object UserActivation]
geolocation:[object Geolocation]
connection:[object NetworkInformation]
plugins:[object PluginArray]
mimeTypes:[object MimeTypeArray]
pdfViewerEnabled:true
webkitTemporaryStorage:[object DeprecatedStorageQuota]
webkitPersistentStorage:[object DeprecatedStorageQuota]
hardwareConcurrency:8
cookieEnabled:true
appCodeName: Mozilla
appName: Netscape
appVersion:5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36
platform: MacIntel
product: Gecko
userAgent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36
language:en-US
...

Client-side objects
Screen Object

- Contains read-only properties describing the display screen and colours.

Property	Description
availHeight	Returns the height of the screen (excluding the Windows Taskbar)
availWidth	Returns the width of the screen (excluding the Windows Taskbar)
colorDepth	Returns the bit depth of the color palette for displaying images
height	Returns the total height of the screen
width	Returns the total width of the screen

Client-side objects
Screen Object (example)

```
//myscript6.js
document.writeln("<h1>Screen Information</h1><p>");
printScreenInfo('width'); printScreenInfo('height');
printScreenInfo('colorDepth'); printScreenInfo('availWidth');
printScreenInfo('availHeight');
function printScreenInfo(name) {
document.writeln("<strong>"+ name + ":</strong>"+ screen[name]+ "<br>");
}
```

Output:

width:1680
height:1050
colorDepth:32
availWidth:1680
availHeight:1016

Client-side objects
History Object (properties)

- The history object contains the URLs visited by the user (within a browser window).
- The history object is part of the window object and is accessed through the window.history property

length – contains the current length of the history list

- The history Object does not contain any values that reflect the actual URLs in the history list.
- Therefore, you cannot, for example, perform some action that reads the value of "URL number 3 in the history list"
- The history Object is designed for navigating the history list.

Client-side objects
History Object (Methods)

- back() – moves the user to the URL one place previous in the history list (previous to the current position).
- forward() - moves the user one URL forward, relative to current position, in the history list
- go(offset) - accepts an integer parameter, positive or negative, as offset.
  - If the parameter is a positive integer, the program will move the user that many places forward in the history list.
  - If the parameter is negative, it will move the user that many places backward (previous to the current position) in the history list.
  - The current position is always place zero.

Client-side objects
History Object (Example)

```
<body>
  <div>
    Go to the page:<br>
    <button onclick="history.go(-1);">Back</button>
    <button onclick="history.go(0);">Current</button>
    <button onclick="history.go(+1);">Forward</button>
  </div>
```

Client-side objects
Location Object (properties)

The location Object contains information on the current URL.

- href - contains the string value of the entire URL
  - window.**open**(location.**href**, "windowName", "feature1,feature2, ..."); open a new window, which connects to the same URL as the current window (so you can look at another portion of the same page at the same time you're looking at the current portion of the page)
  - location.**href** = "http://www.someISP.com/~me/myPage.htm"; Launch a new page without opening a new window (you leave your current page, including other JavaScript programs).

Client-side objects
Location Object (properties)

- host - holds only the *hostname:port* of the current page's URL
  - Example:
    http://www.someISP.com:8080/~me/myPage.htm
    location.host -> " www.someISP.com:8080"
  - Note: Assigning a value to location.host will generate an error because it is nonsensical. Whereas you can assign an entire URL to location.href, which would then connect to that URL. You cannot connect to just a host.
- port - contains the value of the port number in the URL

Client-side objects
Location Object (properties)

- hostname – returns the *hostname* portion of the URL (no port number).
- pathname - is the portion of the URL that describes the location of the Web document on the host computer.
  - Example:
    http://www.someISP.com/~me/myPage.htm
    location.pathname -> /~me/myPage.htm

  Assigning a value to this property,

  location.pathname = "~me/otherdocs/newdoc.htm";

  will cause the Web browser to load that document into the current window.

Client-side objects
Location Object (properties)

- protocol – contains the leftmost portion of the URL.
  - By checking the **location.protocol** property, the JavaScript program can determine if the page currently resides in was delivered by HTTP, FTP, or NEWS.
- hash – contains the value following the hash mark
  - Some URLs contain special hash mark values following the pathname.

    http://www.someISP.com/~me/myPage.htm#item1

    The hash mark (#) specifies the name of an anchor to jump to in the Web page.

    location.hash -> item1

    You can use the location.**hash** property in Event Handlers to bring the user to specific locations within the current page.

Client-side objects
Location Object (properties)

- search – contains the value following the question mark.
  - Some URLs contain search parameters following the pathname, denoted with a question mark (?).
  - A form entry is probably the most common use for a search parameter.

    http://www.someISP.com/~me/myProgram?formData

    location.search -> formData
- reload() – reloads the document that is currently displayed in the window of the location Object.
- assign(newURL) – loads a new document.
- replace(newURL) – replaces the current document with a new one (differently from assign, the user can't go back).

Client-side objects
Location Object (example)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title> Location Object</title>
<script src="./myscript7.js">
</script>
</head>
<body>
<form action="#" method="get">
<p>
<strong>Insert your name:</strong><br>
<input type="text" name="Name">
<input type="button" onclick= "send()" value="SEND">
</p>
</form>
</body>
</html>
```

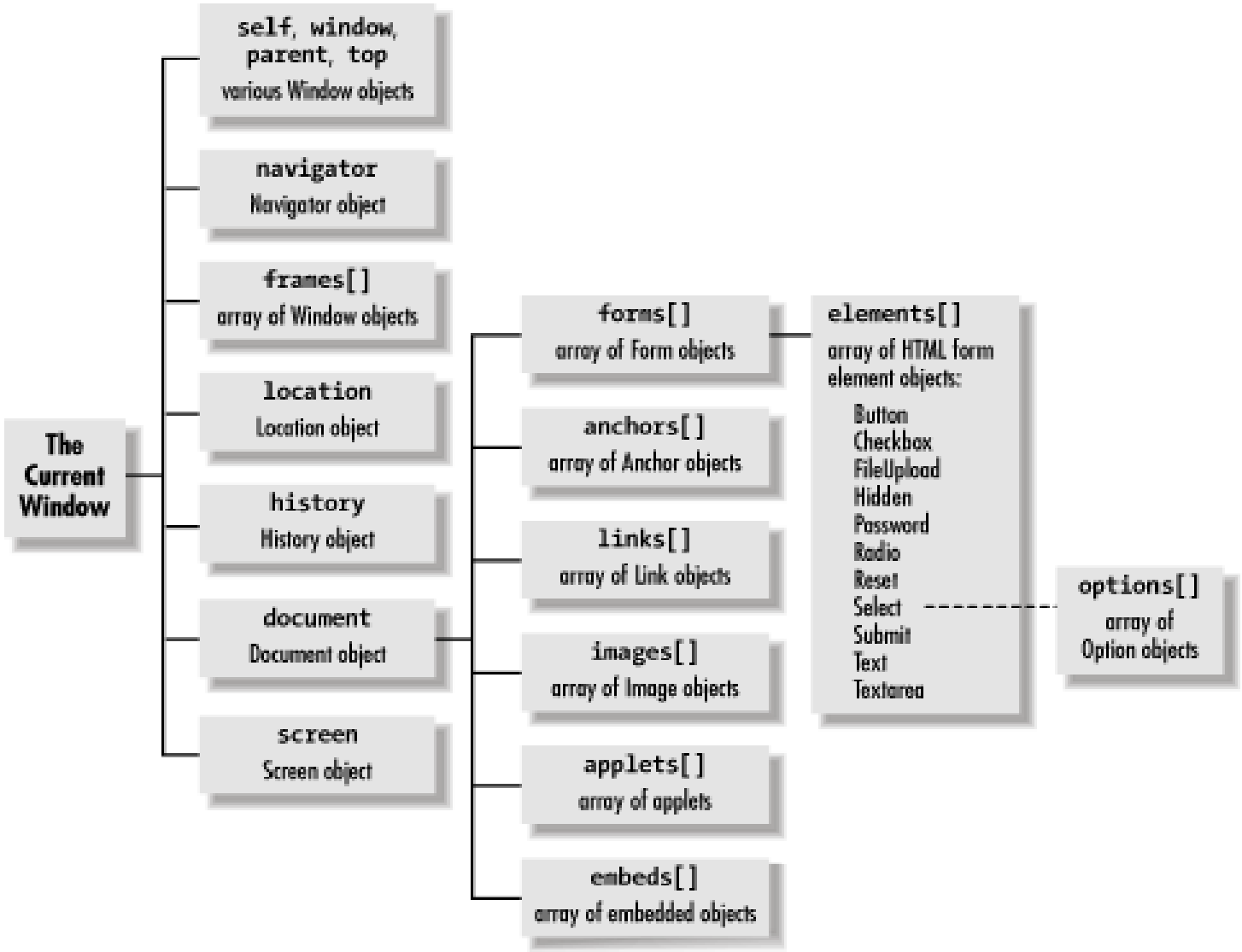
Client-side objects
Location Object (example)

```
display('URL',location.href);
display('Protocol',location.protocol);
display('Host name', location.hostname);
display('Local Address', location.hash);
display('Port', location.port);
display('Path', location.pathname);

function send() {
  location.search="&Name=" + document.forms[0].Name.value;
}

function display(propriety, value) {
  document.writeln("<p><strong>"+propriety+":</strong>"+value+"</p>");
}
```

Client-side objects
Document Object

- Each HTML document loaded into a browser window becomes a Document object.
- The Document object provides access to all HTML elements in a page, from within a script.
- Tip: The Document object is also part of the Window object, and it can be accessed through the **window.document** property
- A document is the file of HTML code that describes a given page.
  - A page is what appears within the browser window. So, every window is associated with a document object.

Client-side objects
Document Object Collections (DOM 0)



Client-side objects
Document Object (Properties)

- **activeElement**: returns the currently focused element
- **body**: returns the body element of the current document
- **cookie**: returns a semicolon-separated list of cookies or sets a single cookie
- **dir**: gets/set direction RTL/LTR
- **head**: returns the head element of the current document
- **location**: returns the URI of the current document
- **title**: gets/sets the title of the document
- **readyState**: the loading status of the document
  - "loading": document is still loading
  - "interactive": loading of document is finished, but sub-resources such as images, fames, and scripts are still loading. We can access the DOM.
  - "complete": page loading is complete.
- **referrer**: the URI of the page from which the user arrived to the current page

Client-side objects
Document Object (Methods)

Method	Description
close()	Closes the output stream previously opened with document.open()
open()	Opens an output stream to collect the output from document.write() or document.writeln()
write()	Writes HTML expressions or JavaScript code to a document
writeln()	Same as write(), but adds a newline character after each statement

NOTE: using these methods is discouraged; writing HTML code using *write()* may lead to bad interactions with the HTML parser; browsers are allowed to not execute JS code introduced via *write()*.
This mechanism has been used in some examples just because we don't know yet how to change the document.

Client-side objects
Document Object (Methods)

- Note: How is it possible to change the presentation style of the elements?
- Use the **style** property.
- Note: the names of the properties do not have hyphens and are in camelCase notation.
- For instance:
  - background-color -> backgroundColor
  - border-right-style -> borderRightStyle
- Example

  element.style.backgroundColor = 'yellow'

Client-side objects
Document Object (Methods)

- write(str) – outputs text to the client window's document
  - document.write() text is parsed into the document model
  - the browser then interprets the HTML and then renders the HTML
- writeln(str) – outputs text to the client window's document (appends a newline character to the end of the output)

Examples:

```
document.write("<h1>Thank you for ordering!</h1>");
document.write(example);
```

where *example* is a string variable

Client-side objects
Document Object (Methods)

- Note: write() method actually takes a variable number of arguments, rather than just one.
  - If more than one argument is given, each of the arguments is interpreted as a string and written in turn.

  document.**write**("This is Part 1 ", "and this is Part 2 ", "and this is Part 3");
- Note: writeln() appends a newline character to the end of the output.
  - Keep in mind that these methods output their parameters as HTML. HTML ignores newline characters when it comes to outputting to the screen.
  - HTML does not insert line breaks in screen output unless you specify a line break using either
 or <p> tags or text resides between <pre> and </pre> tags.

Client-side objects
Document Object (Example)

```
<head>
<meta charset="utf-8">
<title>
Document Object
</title>
<script src="./myscript8.js">
</script>
<style>
a:link { color: blue;}
/* visited link */
a:visited { color: red;}
/* mouse over link */
a:hover { color: orange;}
/* selected link */
a:active { color: green;}
</style>
</head>
```

Client-side objects
Document Object (Example)

```
<body>
<p>
<a href="#" onclick="openW('yellow')">open yellow window</a><br>
<a href="#" onclick="openW('red')">open red window</a><br>
<a href="#" onclick="openW('blue')">open blue window</a>
</p>
<p>
<a href="#" onclick="openDoc('yellow')">open yellow page</a><br>
<a href="#" onclick="openDoc('red')">open red page</a><br>
<a href="#" onclick="openDoc('blue')">open blue page</a>
</p>
</body>
```

```
// myscript8.js
document.writeln("<p>You are on '"+ document.URL + "'<p>");

function openW(color) {
  win = window.open("",color,"width=400,height=250");
  HTMLcode = '<html><head><meta charset="utf-8"><title>Page '+
                    color+ '</title></head>' +
          '<body><p>This is a ' + color +
          ' page </p></body></html>';
  win.document.writeln(HTMLcode);
  win.document.body.style.background = color;
  win.document.body.style.color = "white";
}
function openDoc(color) {
  doc = document.open("text/html","replace");
  HTMLcode = '<html><head><meta charset="utf-8"><title>Page '+
          color+ '</title></head>' +
          '<body><p>This is a ' + color +
          ' page </p></body></html>';
  doc.writeln(HTMLcode);
  doc.body.style.backgroundColor = color;
  doc.body.style.color = "green";
  doc.close();
}
```

Client-side objects
Document Object (Methods)

Double and Single Quotes

- When writing HTML it is general practice to use double quotes for tag attributes, e.g.:

```
<img src="picture.gif" width="400" height="400">
```

- When writing JavaScript it is general practice to use single quotes for string literals:

```
<script>
  document.write('<p>');
</script>
```

In general these quoting styles then complement one another when outputting HTML using JavaScript

Client-side objects
How are objects named? (DOM 0)

- You can access only to a subset of all elements contained in a document, through the document object.
- The name of each object is prefaced by the names of all the objects that contain it, in order, separated by dots.
- For example, the window Object contains the document Object, and the document Object contains a form Object, and the form Object contains text input fields, buttons, select Objects, etc...

window.document.forms[0].elements[3]

Client-side objects
How are objects named? (DOM 0)

Leave off the word "window."

- All Client-side Object names officially start with the highest Object name, "window". But since all names start with "window", JavaScript allows you to leave that off.
  - **document.forms[0].elements[3]**

Client-side objects
How are objects named? (DOM 0)

Form elements

- The numerous Object types included in the Forms Array are collectively referred to as "elements" and they can all be referred to by means of the "Elements Array" (e.g.: elements[1]).

- Example: **document.forms[0].elements[1]**

Client-side objects
How to refer to properties or methods?

- Note: JavaScript is case sensitive
  - **document.myform.Check1** is not the same object as **document.myform.check1**
- The **form** object has child objects named **button1** and **text1**, corresponding to the button and text field in the form.
- These objects have their own properties based on their HTML attribute values, for example,
  - *document.myform.button1.value* is "Press Me"
  - *document.myform.elements[2].name* is "Button1"
  - *document.myform.elements[0].name* is "text1"

Client-side objects
How to refer to properties or methods?

- Three different ways to specify the value of an object:
  - **document.myform.text1.value** is "blahblah"
    NOTE: using just the object names
  - **document.myform.elements[0].value** is "blahblah"
    NOTE: using an object name & array notation (to specify the object instead of the object name)
  - **document.forms[0].elements[0].value** is "blahblah"
    NOTE: using just array notation

Client-side objects
How to refer to properties or methods?

```
<title>A Simple Document</title>
</head>
<body>
<p><a name="top" id="top">This is the top of the page</a><p>
<hr>
<form method="post" action="mailto:nobody@dev.null">
<p>Enter your name: <input type="text" name="me" size="70">
<input type="Submit" value="OK">
<input type="Reset" value="Oops"></p>
</form>
<hr>
<p>Click here to go to the <a href="#top">top</a> of the page</p>
</body>
</html>
```

Client-side objects
How to refer to properties or methods?

- The code creates an HTML page with an anchor at the top of the page and a link to that anchor at the bottom.
- In between is a simple form that allows the user to enter his name.
- We can also access the other HTML elements of this document using the following properties:
  - anchors
  - forms
  - links

Client-side objects
How to refer to properties or methods?

document.title

<title>A very simple HTML page</title>

document.anchors[0]

This is the top of the page

document.anchors[0].name -> top

document.forms[0]

<form method= "post" action="mailto:nobody@dev.null">

document.forms[0].method -> post

document.forms[0].action -> mailto:nobody@dev.null

Client-side objects
How to refer to properties or methods?

document.forms[0].elements[0]

<input type="text" name="me" size="70">

document.forms[0].elements[0].name -> me

document.forms[0].elements[0].type -> text

document.forms[0].elements[1]

<input type="Submit" value="OK">

document.forms[0].elements[1].value -> OK

document.forms[0].elements[1].type -> Submit

document.forms[0].elements[2]

<input type="Reset" value="Oops">

document.forms[0].elements[2].value -> Oops

document.forms[0].elements[2].type -> Reset

Client-side objects
How to refer to properties or methods?

document.links[0]

top

document.links[0].href

file:///F:/Examples/example1.htm#top

Client-side objects
Document

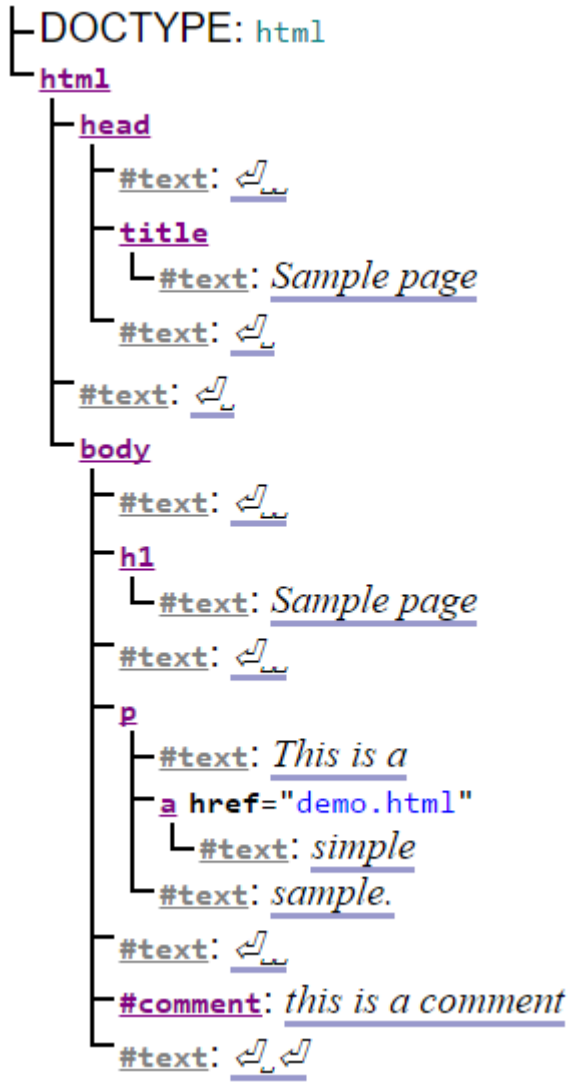
- Each object in the document has specific properties and methods.
- The access to each element using the approach described so far is not flexible, is limited and does not allow changing dymamically the structure of the document.
- The ability to change a Web page dynamically with a scripting language is made possible by the Document Object Model (DOM), which can connect any element on the screen to a JavaScript function.

Document Object Model
DOM

- "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
- The DOM is separated into different parts / levels:
  - Core DOM - standard model for any structured document
  - HTML DOM - standard model for HTML documents
- The DOM defines the objects and properties of all document elements, and the methods (interface) to access them.

Document Object Model
DOM

```
<!DOCTYPE html>
<html>
 <head>
  <title>Sample page</title>
 </head>
 <body>
  <h1>Sample page</h1>
  <p>This is a <a href="demo.html">simple</a>
sample.</p>
  <!-- this is a comment -->
 </body>
</html>
```



DOM

The HTML DOM is:

- A standard object model for accessing and manipulating HTML documents
  - A standard programming interface for HTML
  - Platform- and language-independent
  - A W3C standard

- In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

DOM

- According to the DOM, everything in an HTML document is a **node**.
  - The entire document is a ***document node***
  - Every HTML element is an ***element node***
  - The text in the HTML elements are ***text nodes***
  - Every HTML attribute is an ***attribute node***
  - Comments are ***comment nodes***

DOM

```
<html>
  <head>  <title>DOM Tutorial</title>  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

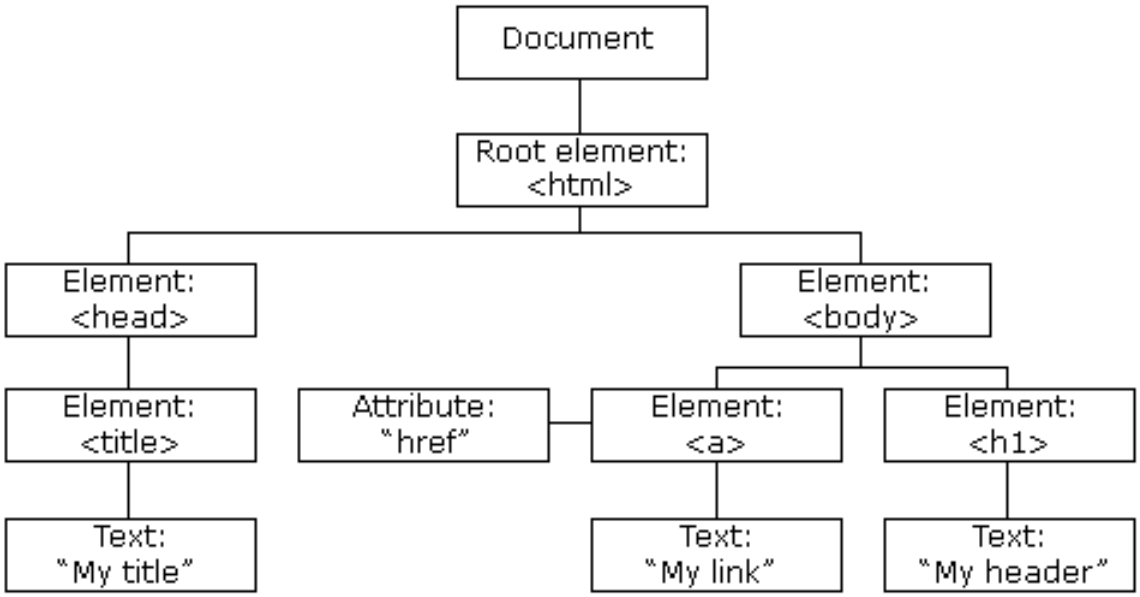
- The root node in the HTML above is <html>. All other nodes in the document are contained within <html>.
- The <html> node has two child nodes: <head> and <body>.
- The <head> node holds a <title> node. The <body> node holds a <h1> and <p> nodes.

DOM

- A common error in DOM processing is to expect an element node to contain text. However, the text of an element node is stored in a text node.
- In this example: <title>DOM Tutorial</title>, the element node <title> holds a text node with the value "DOM Tutorial".
  - "DOM Tutorial" is not the value of the <title> element

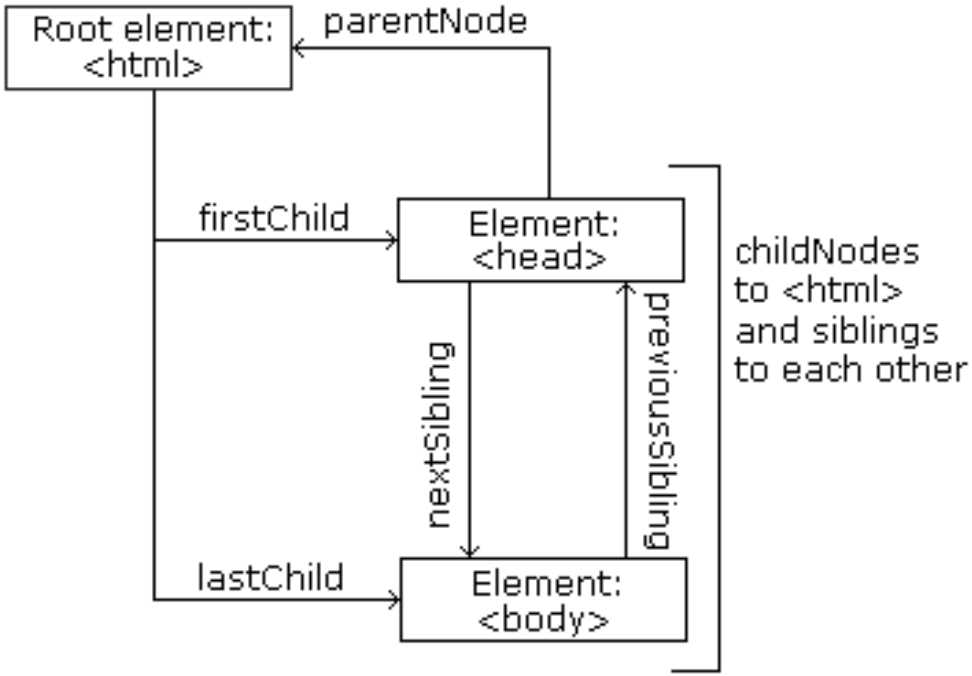
DOM Tree

- The HTML DOM views a HTML document as a tree-structure. The tree structure is called a node-tree.
- All nodes can be accessed through the tree.



DOM Tree

- Parent nodes have children.
- Children on the same level are called siblings (brothers or sisters).
- Every node, except the root, has exactly one parent node



DOM Tree

- <head> element: first child of the <html> element
- <body> element: last child of the <html> element
- <h1> element: first child of the <body> element
- <p> element: last child of the <body> element

- The programming interface of the DOM is defined by standard properties and methods.

DOM (Level 3) Tree

- Some attributes:
  - x.innerHTML — the HTML content of x (not recommended)
  - x.childNodes — the child nodes of x
  - x.parentNode — the parent of x
  - x.firstChild — the first child of x
  - x.lastChild — the last child of x
  - x.nodeName — the name of x
  - x.nodeValue — the value of x
  - x.nodeType — the type of x
  - x.nextSibling — next brother of x
  - x.previousSibling — previous brother of x
  - x.textContent — the text content of x and its descendants
  - x.attributes — the attributes nodes of x
- Note: In the list above, x is a node object (HTML element).

DOM (Level 3) Tree

- Why is not x.innerHTML recommended?
  - **The specified value is parsed as HTML**
  - Thus

```
username = "<script>alert('I am an annoying alert!')</script>";
el.innerHTML = username;
```

The code is executed (possible attacks)

For that reason, it is recommended that you do not use *innerHTML* when inserting plain text; instead, use Node.textContent. This doesn't parse the passed content as HTML, but instead inserts it as raw text.

Warning: If your project is one that will undergo any form of security review, using *innerHTML* most likely will result in your code being rejected

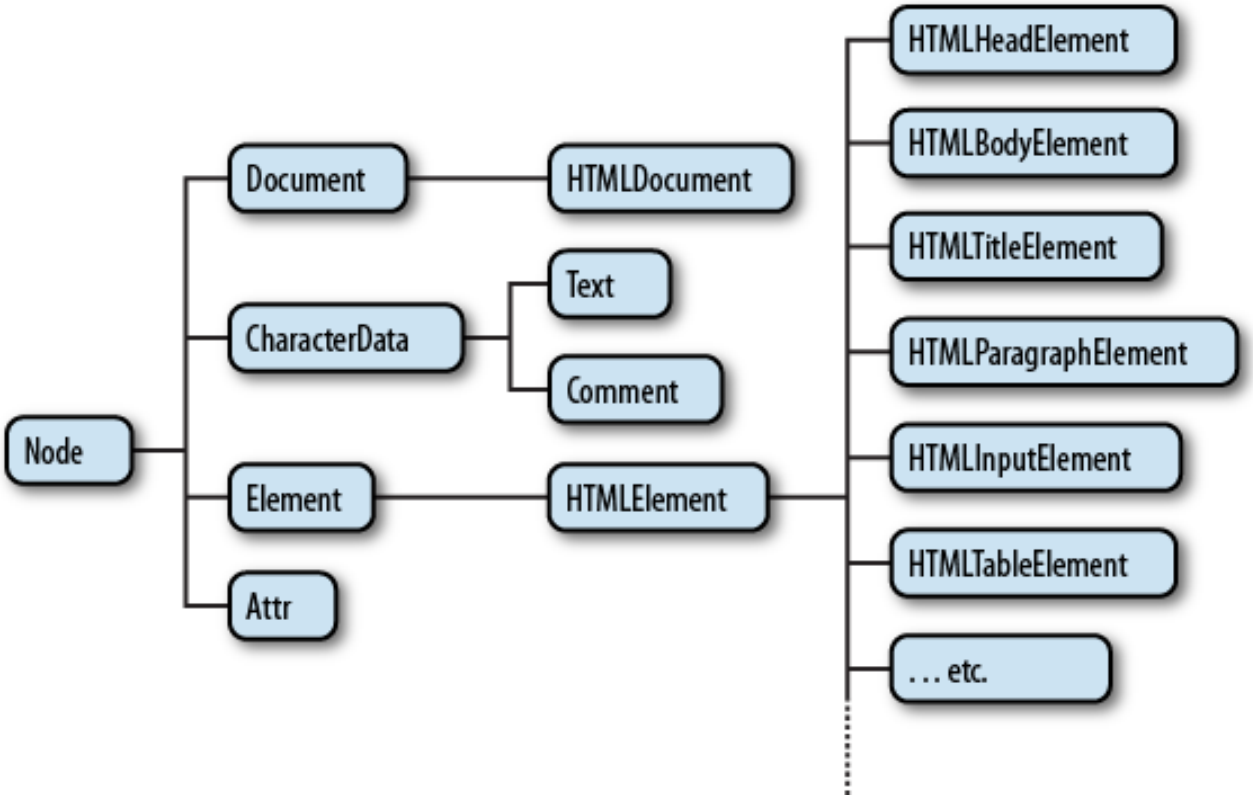
DOM (Level 3) Tree

- **myNode.nodeType == Node.ELEMENT_NODE**
  myNode is an object Element
- **myNode.nodeType == Node.ATTRIBUTE_NODE**
  myNode is an attribute

Main types of nodes:
const unsigned **short** ELEMENT_NODE = 1
const unsigned **short** ATTRIBUTE_NODE = 2
const unsigned **short** TEXT_NODE = 3
const unsigned **short** ENTITY_NODE = 6 (XML documents)
const unsigned **short** PROCESSING_INSTRUCTION_NODE = 7
const unsigned **short** COMMENT_NODE = 8
const unsigned **short** DOCUMENT_NODE = 9

DOM (Level 3) Tree

As specified by the DOM API, each HTML element is represented by an object, hence the term “Object Model”.



DOM (Level 3) Tree (Methods)

Finding, adding, manipulating nodes

- x.getElementById(id) gets the element with the specified id
- x.getElementsByTagName(name) gets all elements with a specified tag name
- x.appendChild(node) adds a child node to the end of the list of children of x
- newx = x.cloneNode(deep) clones x to create a new object: if deep == true, all the tree with x as root is cloned; otherwise only node x
- x.removeChild(node) removes a child node from x and returns it

DOM (Level 3) Tree (Methods)

- x.replaceChild(newC, oldC) replaces oldC with newC and returns oldC
- x.hasChildNodes() returns true if the object has children; false otherwise
- x.contains(node) returns a Boolean value indicating whether a node is a descendant of x or not
- x.isEqualNode(node) tests whether x is equal to node

DOM (Level 3) Tree (Methods)

Read and set the attributes of elements

- x.getAttribute(string_name) – returns the value of the attribute string_name
- x.setAttribute(string_name, string_value) – set the value of the attribute string_name to string_value

```
<script>
function showAttributes() {
  let result;
  let elems = document.getElementsByTagName("input");
  for (let i=0;i<elems.length;i++) {
    result = "";
    for (let a in elems.item(i)) {
      aValue = elems.item(i).getAttribute(a);
      if (a=='type')                                    // If the attribute is type
        result += "- " + a + ": " + aValue + "\n";
    }
    window.alert("Types of " + elems.item(i).tagName + ":\n" + result);
  }
}
</script>
```

DOM (Level 3) Tree (Methods)

- **getElementsByTagName()** returns a HTMLcollection of all elements with a specified tag name
- **HTMLCollection** is an array-like collection of HTML elements. The elements in a collection can be accessed by index using array-like notation or the item() method.

```
<body>
<div>
<input type="text" size="20"><br>
<input type="text" size="20"><br>
<input type="text" size="20"><br><br>
<input type="button" onclick="showAttributes()"
value="Would you like to show the type of the input elements?">
</div>
</body>
```

Three elements of type text and one of type button

DOM Tree (get or modify the content of an element)

- Attention: The easiest way to get or modify the content of an element is by using the innerHTML property, but it is **not recommended**.
- Solution: Use the **childNodes** and **nodeValue** properties

```
<script>
function change() {
  let txt = document.getElementById("intro").childNodes[0].nodeValue;
  console.log("The value of the intro paragraph: " + txt);
  document.getElementById("intro").childNodes[0].nodeValue = "Hi, some new text";
}
function askAndChange(){
  let r = confirm("Change the text!");
  if (r === true)
    change();
}
</script>
</head>
<body>
<p id="intro">Hello World!</p>
<input type="button" onclick="askAndChange()" value="Change text">
</body>
```

DOM Tree (get or modify the content of an element)

- Other solution: use the **firstChild** and **nodeValue** properties

```
function change() {
  let txt = document.getElementById("intro").firstChild.nodeValue;
  console.log("The value of the intro paragraph: " + txt);
  document.getElementById("intro").firstChild.nodeValue = "Hi, some new text";
}
```

DOM Tree
Access to a node

You can access a node in three ways:

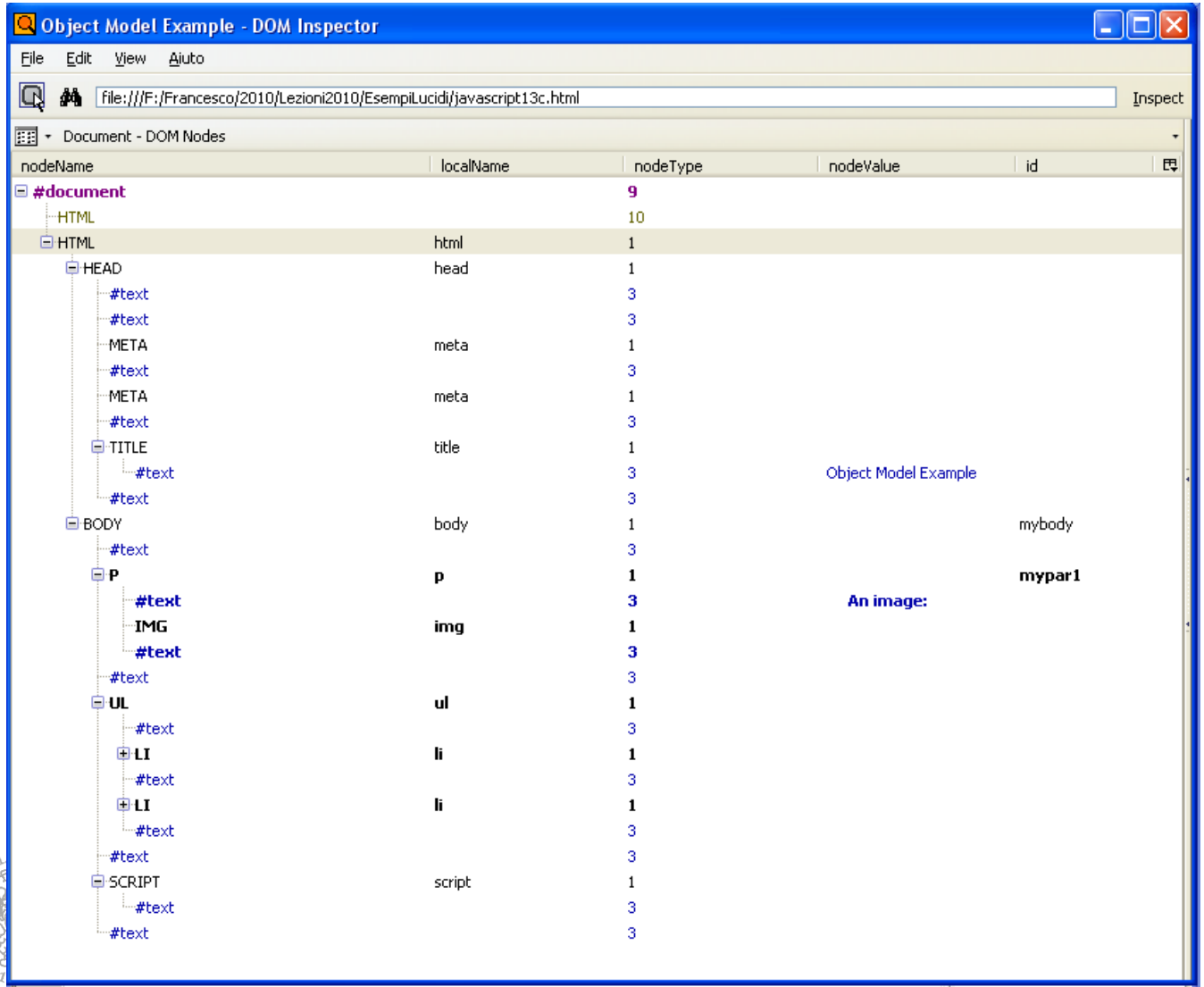
1. By using the getElementById() method
2. By using the getElementsByTagName() method
3. By navigating the node tree, using the node relationships

DOM Tree
Access to a node

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title> Object Model Example </title>
</head>
<body id="mybody">
<p id="mypar1">
An image:<img src="planets.gif"
height=30 width=30 alt="planets">
</p>
<ul>
<li>First</li>
<li>Second</li>
</ul>
<script src="./myscript1.js"></script>
</body>
</html>
```

```
let text = document.getElementById("mypar1").firstChild.nodeValue;
let type = document.getElementById("mypar1").firstChild.nodeType;
window.alert("The paragraph has the text " + text);
window.alert("The type of the node is " + type);
```

DOM Tree
Access to a node



DOM Inspector

text nodes corresponding to spaces and newlines

DOM Tree
Access to a node

2. Using the getElementsByTagName() method (file myscript2.js)

```
let elems = document.getElementsByTagName("p");
let text = elems[0].firstChild.nodeValue;
window.alert("The paragraph has the text " + text);
```

DOM Tree
Access to a node

3. Navigating the DOM tree (example 1 - file myscript3.js)

```
function visit(node, obj, depth) {
  obj.txt += ".".repeat(depth) + node.nodeName +
             ", type: " + node.nodeType +
             ", value: " + node.nodeValue+ "<br>";
  if (node.hasChildNodes()) {
    for (let i=0; i<node.childNodes.length; i++) {
      visit(node.childNodes[i], obj, depth + 1);
    }
  }
}

let node = document.documentElement; // this is the <html> element
let obj = new Object();
obj.txt = "";
visit(node, obj, 0);
document.body.style.fontFamily = "Courier";
document.write(obj.txt);
```

DOM Tree
Access to a node

3. Navigating the DOM tree (example 2 - file myscript4.js)

```
function visit(node, obj, depth) {
  if (node == null) return;
  obj.txt += ".".repeat(depth) + node.nodeName +
            ", type: " + node.nodeType +
            ", value: " + node.nodeValue+ "<br>";
  if (node.firstChild) {
    visit(node.firstChild, obj, depth+1);
  }
  if (node.nextSibling) {
    visit(node.nextSibling,obj, depth);
  }
}

let node = document.documentElement;
let obj = new Object();
obj.txt = "";
visit(node,obj, 0);
document.body.style.fontFamily = "courier";
document.write(obj.txt);
```

Events

Event Object (Definition)

- Every element on a web page may generate events which can trigger a JavaScript handler.
- The event object gives you information about an event that has occurred.
- The **Event** object represents the state of an event, such as the element in which the event occurred, the status of the keyboard keys, the location of the mouse, and the status of mouse buttons.
- Events are normally used in combination with functions (Event **handlers**), and the function will not be executed before the event occurs.
- The event object contains properties that describe a JavaScript event, and is passed as an argument to an event handler when the event occurs.

Event Object (Definition)

Example

- mouse-down event. The event object contains
  - the type of event (in this case **MouseDown**),
  - the **x** and **y** position of the cursor at the time of the event,
  - a number representing the mouse button used,
  - a field containing the modifier keys (Control, Alt, Meta, or Shift) that were depressed at the time of the event.
- The properties used within the event object vary from one type of event to another.

Event Object
To Associate Events with Code

- **First approach** – Events as HTML element attributes
  - The event is an attribute of an HTML element
    <tag eventName = "JavaScript code or call to handler">
    eventName starts with **on**
  - Example:
    <body onload = "myHandler()"
    <body onload = "instruction1;…; instructionN;">

Note: it is not possible to associate any event with any element since some events have no meaning or cannot arise for some element.

Event Object
To Associate Events with Code

```
<body>
<div style="position:absolute; left:300px; top:200px;" id="but">
<button onmouseover="moveMouse()">Click</button>
</div><script src="./myscript10.js">
</script>
</body>
```

Event Object
To Associate Events with Code

```
const but = document.getElementById('but');
const diff = [150, 0, -150, 0];
let i = 0;
function moveMouse() {
  but.style.top= (parseInt(but.style.top)+ diff[(i+3)%4]) + "px";
  but.style.left=(parseInt(but.style.left)+ diff[(i++)%4])+ "px";
}
```

Event Object
To Associate Events with Code

- **Second Approach** – set the value of the property corresponding to the event to a specific function.

```
<body>
<div style="position:absolute; left:300px; top:200px;"
id="but">
<button >Click</button>
</div>
<script src="./myscript11.js"> </script>
</body>
```

```
//myscript11.js
const but = document.getElementById('but');
const diff = [150, 0, -150, 0];
var i = 0;
but.onmouseover = moveMouse;
function moveMouse() {
  but.style.top= (parseInt(but.style.top)+ diff[(i+3)%4]) + "px";
  but.style.left=(parseInt(but.style.left)+ diff[(i++)%4])+ "px";
}
```

Note: no parentheses

Event Object
To Associate Events with Code

- Third Approach – Specifying an event handler with a *Function* object.

  Function objects created with the Function constructor are evaluated each time they are used.

```
function dettaglia(elemento, evento) {
  console.log(elemento + ": " + evento);
  console.log("\t name=" + elemento.name + ", type=" + elemento.type);
  if(elemento.type=="text" && evento=="Change") {
    console.log("\t the new value of the text field is " + elemento.value);
  }
}
```

Event Object
To Associate Events with Code

- **Third Approach** – Specifying an event handler with a Function object.

```
// La funzione aggiungi_gestori installa un insieme di gestori di
// eventi su ogni elemento di un form f, senza controllare se
// l'elemento supporta tutti questi tipi di eventi

function aggiungi_gestori(f) {
  const gestore_click = new Function("dettaglia(this, 'Click')");
  const gestore_change = new Function("dettaglia(this, 'Change')");
  const gestore_focus = new Function("dettaglia(this, 'Focus')");
  const gestore_blur = new Function("dettaglia(this, 'Blur')");
  const gestore_select = new Function("dettaglia(this, 'Select')");
  const gestore_dblclick = new Function("dettaglia(this, 'DblClick')");
  for(let i = 0; i < f.elements.length; i++) {
    const e = f.elements[i];
    e.onclick = gestore_click;
    e.onchange = gestore_change;
    e.onfocus = gestore_focus;
    e.onblur = gestore_blur;
    e.onselect = gestore_select;
    e.ondblclick = gestore_dblclick;
  }
}
const f = document.getElementById("f1");
aggiungi_gestori(f);
```

Event Object
To Associate Events with Code

```
<body>
<form id=f1>
<input type=text name=first_field><br>
<input type=text name=second_field><br>
<input type=button name=first_button value=PRESS><br>
<input type=button name=second_button value=CANCEL>
</form>
<script src="./myscript-eventi.js">
</script>
</body>
```

Event Object
To Associate Events with Code

- **Fourth approach** – addEventListener(type, listener[, useCapture])
- The EventTarget.addEventListener() method registers the specified listener on the EventTarget it is called on. The event target may be an Element in a document, the Document itself, a Window, or any other object that supports events
  - type A string representing the event type to listen for
  - listener The object that receives a notification when an event of the specified type occurs
  - useCapture If true, useCapture indicates that the user wishes to initiate capture. After initiating capture, all events of the specified type will be dispatched to the registered listener before being dispatched to any EventTarget beneath it in the DOM tree.

Event Object
To Associate Events with Code

- **Fourth approach** – addEventListener(type, listener[, useCapture])

```
// Function to change the content of t2
function modifyText() {
  var t2 = document.getElementById("t2");
  if (t2.firstChild.nodeValue == "three") {
    t2.firstChild.nodeValue = "two";
  } else {
    t2.firstChild.nodeValue = "three";
  }
}
// add event listener to table
var el = document.getElementById("outside");
el.addEventListener("click", modifyText, false);
```

Events

- Main events

Attribute	The event occurs when...
onblur	An element loses focus
onchange	The content of a field changes
onclick	Mouse clicks an object
ondblclick	Mouse double-clicks an object
onerror	An error occurs when loading a document or an image
onfocus	An element gets focus
onkeydown	A keyboard key is pressed

Events

Attribute	The event occurs when...
onkeypress	A keyboard key is pressed or held down
onkeyup	A keyboard key is released
onmousedown	A mouse button is pressed
onmousemove	The mouse is moved
onmouseout	The mouse is moved off an element
onmouseover	The mouse is moved over an element
onmouseup	A mouse button is released
onresize	A window or frame is resized
onselect	Text is selected
onunload	The user exits the page

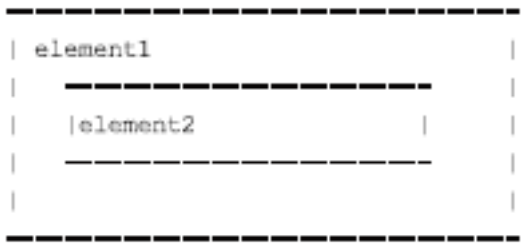
Events

- The events are applicable to specific elements
- For instance,
  - onblur is applicable to only object window and all the elements of a form
  - onchange is applicable to only the fields <input>, <textarea> and <select> of a form
  - onmouseout is applicable to elements, links, and areas
  - onresize is applicable to document, frame, and window

Events

```
<head>
<meta charset="utf-8">
<title>Example</title>
<script >
function setStyle(x) {
  document.getElementById(x).style.background="yellow";
}
function resetStyle(x) {
  document.getElementById(x).style.background="white";
}
</script>
</head>
<body>
<div>
First name: <input type="text" onfocus="setStyle(this.id)"
onblur="resetStyle(this.id)" id="fname">
<br>
Last name: <input type="text" onfocus="setStyle(this.id)"
onblur="resetStyle(this.id)" id="lname">
</div>
</body>
```

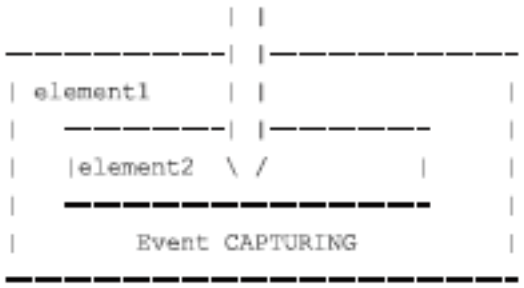
Event order



Two models

- **Capturing** – the event on element1 takes place first (Netscape)
- **Bubbling** – the event on element2 takes precedence (Microsoft)

Event order



Event **capturing**

- The event handler of element1 fires first, the event handler of element2 fires last

Event order

Event **bubbling**

- The event handler of element2 fires first, the event handler of element1 fires last

Event order

W3C model

- W3C has decided to take a middle position in this struggle. Any event taking place in the W3C event model is first captured until it reaches the target element and then bubbles up again

Event Handler

- The Web developer can choose whether to register an event handler in the capturing or in the bubbling phase.
  - This is done through the **addEventListener()**
  - If its last argument is **true** the event handler is set for the **capturing** phase
  - If it is **false** the event handler is set for the **bubbling** phase (default value)

Event Handler

```
element1.addEventListener('click',doSomething2,true)
element2.addEventListener('click',doSomething,false)
```

If the user clicks on element2 the following happens:

1. The `click` event starts in the capturing phase. The event looks if any ancestor element of element2 has a `onclick` event handler for the capturing phase.
2. The event finds one on element1. `doSomething2()` is executed.
3. The event travels down to the target itself, no more event handlers for the capturing phase are found. The event moves to its bubbling phase and executes `doSomething()`, which is registered to element2 for the bubbling phase.
4. The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase. This is not the case, so nothing happens.

Event Handler

```
element1.addEventListener('click',doSomething2,false)
element2.addEventListener('click',doSomething,false)
```

Now if the user clicks on element2 the following happens:

1. The `click` event starts in the capturing phase. The event looks if any ancestor element of element2 has a `onclick` event handler for the capturing phase and doesn't find any.
2. The event travels down to the target itself. The event moves to its bubbling phase and executes `doSomething()`, which is registered to element2 for the bubbling phase.
3. The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase.
4. The event finds one on element1. Now `doSomething2()` is

Event Object Properties

- When an event occurs, the browsers make an **Event object** available to the handlers
- The object provides information regarding the event.

- If you want to stop the propagation:

  stopPropagation() can be called on the event object

- Stopping propagation does not prevent default action (e.g. clicks on links are still processed)

  preventDefault() can be called on the event object to prevent the default action.

Event Handler

- Current Target
  - During the capturing and bubbling phases the target does not change: it always remains a reference to element2

```
element1.onclick = doSomething;
element2.onclick = doSomething;
```

  - How do you know which HTML element is currently handling the event?
  - **target** is not useful, it always refers to **element2**
  - W3C added **currentTarget**
  - You can however use the **this** keyword in the event handler.

An example with capture

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>Example about capture</title>
 <script src="./propagazione-js.js"></script>
 <style>
  #mydiv {background-color: lightblue; width: 20em; height: 20em;}
  #myul {background-color: mintcream; width: 15em; height: 15em;}
  li {background-color: coral; margin: 1em;}
 </style>
</head>
<body onload="letsgo()">
 <div id="mydiv">
  Some text before the list.
  <ul id="myul">
   <li id="myfirstli">First list item</li>
   <li id="mysecondli">Second list item</li>
  </ul>
 </div>
 <label for="capture">Capture</label>
 <input type="checkbox" id="capture">
</body>
</html>
```

An example with capture

```
// References to the elements
let dive;
let ule;
let lie;
let ce;

// Capturing
let capture = false;

// Retrieve the references to the elements
// Set the listener for the checkbox and the li elements
// Set the listener for the div and ul, in capture or not
function letsgo(){
  dive = document.querySelector("div");
  ule = document.querySelector("ul");
  lie = document.querySelectorAll("li");
  ce = document.querySelector('input[type="checkbox"]');

  ce.addEventListener("click", captureSwitch);
  for(let i of lie)
    i.addEventListener("click", liHandler);
  setListeners(capture);
}
```

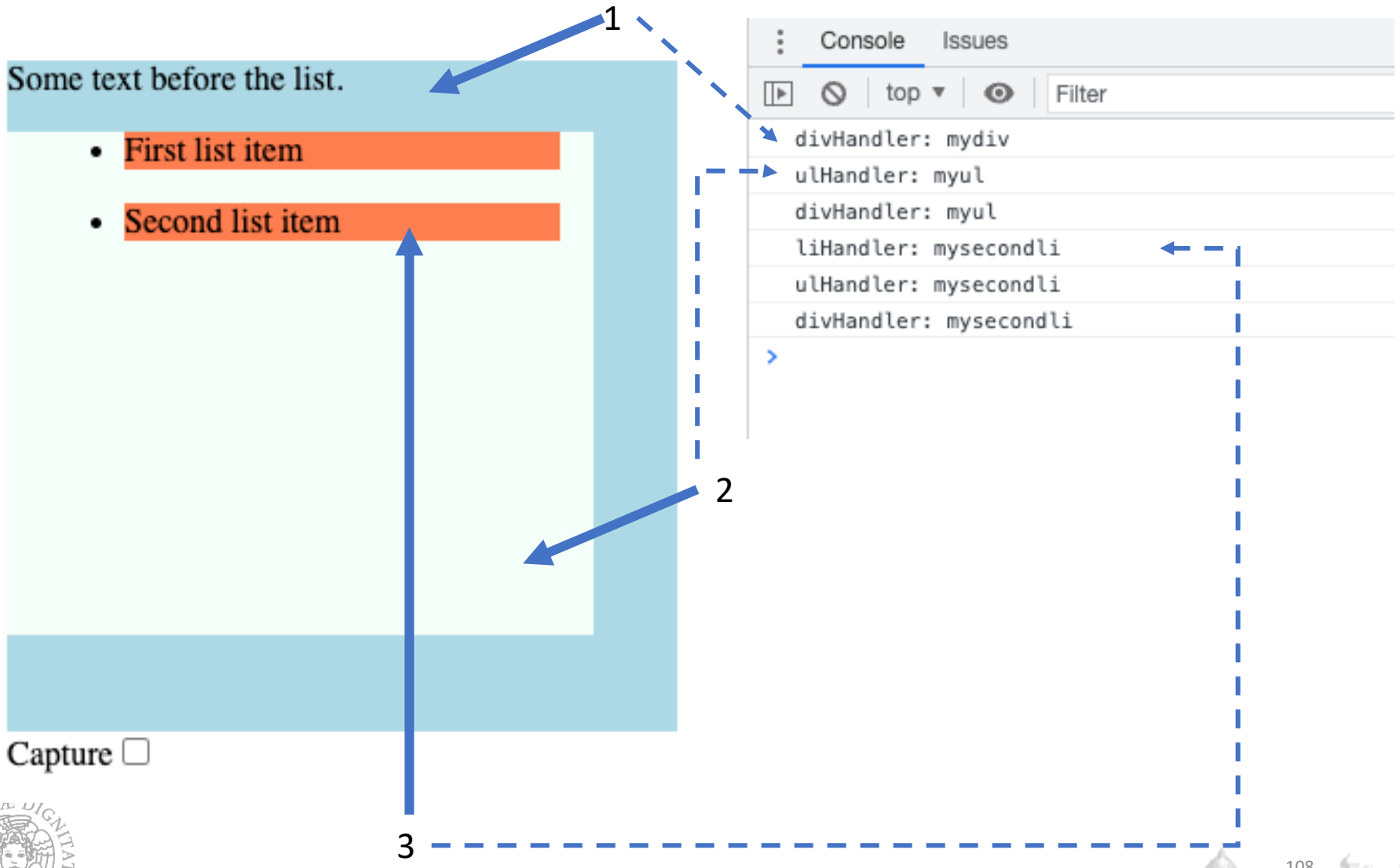
An example with capture

```
// Set the listeners for the div and the ul
function setListeners(iWantToCapture) {
  dive.addEventListener("click", divHandler, iWantToCapture);
  ule.addEventListener("click", ulHandler, iWantToCapture);
}
// Remove the listeners for the div and the ul so that thay can be
// added again in a different mode
function removeListeners(iWantToCapture) {
  dive.removeEventListener("click", divHandler, iWantToCapture);
  ule.removeEventListener("click", ulHandler, iWantToCapture);
}
// Handler for the div
// It stops propagation
// In capture mode it is invoked as the first one and the event is
// not propagated to the contained element
function divHandler(event) {
  console.log("divHandler: " + event.target.id);
  event.stopPropagation();
}
```

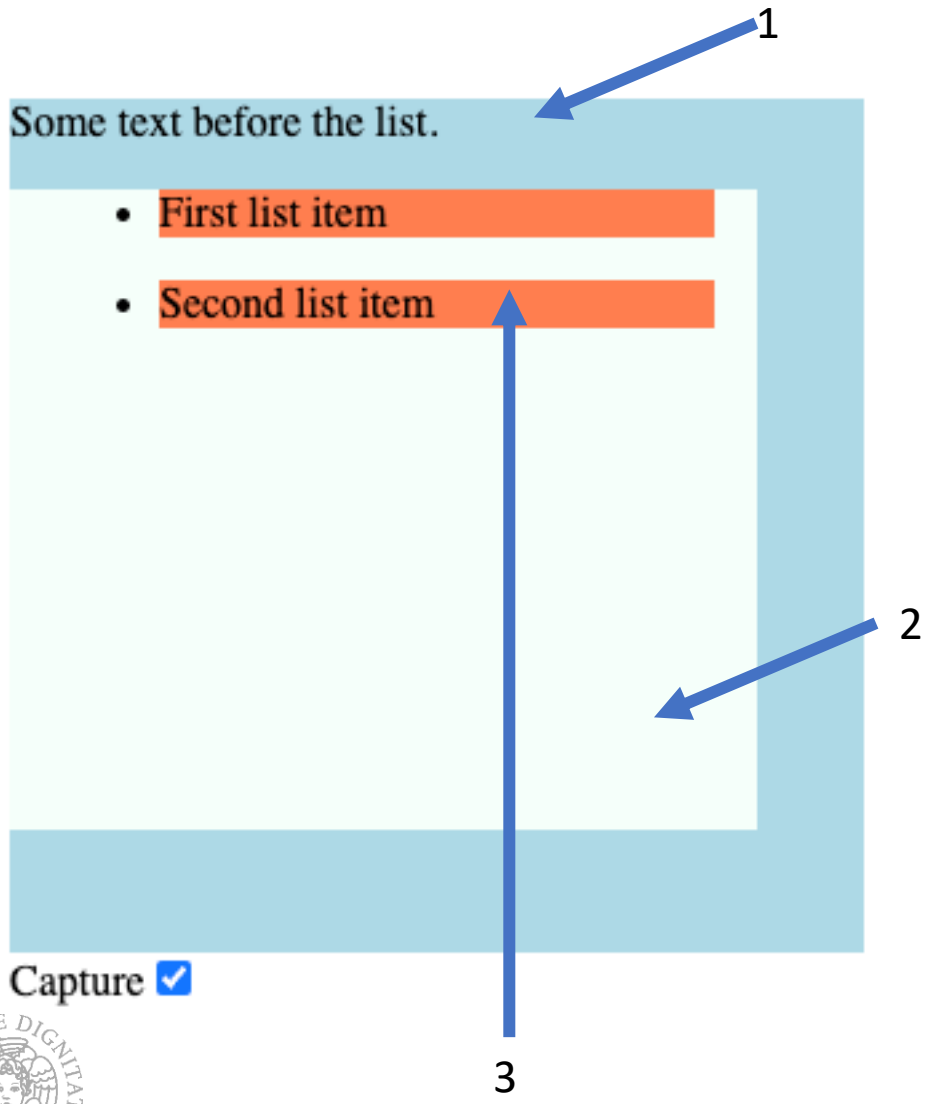
An example with capture

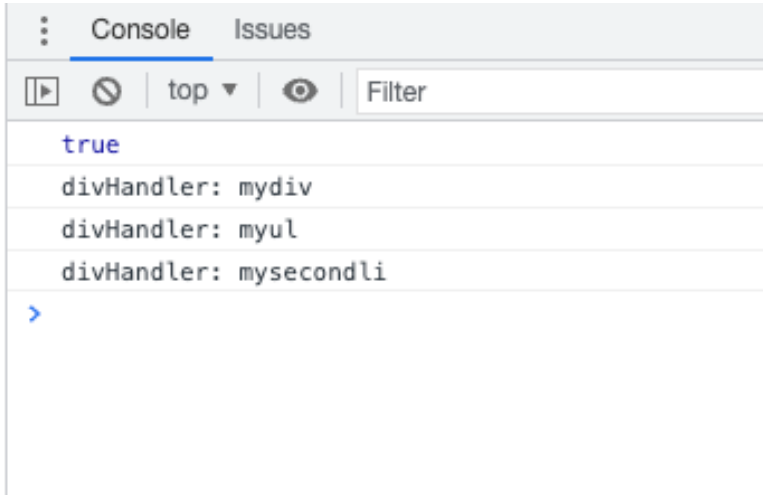
```
// Handler for the ul
function ulHandler(event) {
  console.log("ulHandler: " + event.target.id);
}
// Handler for the li
function liHandler(event) {
  console.log("liHandler: " + event.target.id);
}
// Handler for the checkbox
function captureSwitch() {
  console.log(ce.checked);
  // To remove the listeners the old value of checked is used
  removeListeners(!ce.checked);
  // Set the listener again, but in a different mode
  setListeners(ce.checked);
}
```

An example with capture



An example with capture





Event Object Properties

Property	Description
altKey	Returns whether or not the "ALT" key was pressed when an event was triggered
button	Returns which mouse button was clicked when an event was triggered
clientX	Returns the horizontal coordinate of the mouse pointer when an event was triggered
clientY	Returns the vertical coordinate of the mouse pointer when an event was triggered
ctrlKey	Returns whether or not the "CTRL" key was pressed when an event was triggered
metaKey	Returns whether or not the "meta" key was pressed when an event was triggered
relatedTarget	Returns the element related to the element that triggered the event
screenX	Returns the horizontal coordinate of the mouse pointer when an event was triggered
screenY	Returns the vertical coordinate of the mouse pointer when an event was triggered
shiftKey	Returns whether or not the "SHIFT" key was pressed when an event was triggered

Event Object Properties

Property	Description
bubbles	Returns a Boolean value that indicates whether or not an event is a bubbling event
cancelable	Returns a Boolean value that indicates whether or not an event can have its default action prevented
currentTarget	Returns the element whose event listeners triggered the event
eventPhase	Returns which phase of the event flow is currently being evaluated
target	Returns the element that triggered the event
timeStamp	Returns the time stamp, in milliseconds, from the epoch (system start or event trigger)
type	Returns the name of the event
isTrusted	Indicates whether or not the event was initiated by the browser or by a script

Event Object
Events

Event Handlers can be used in three different ways to trigger a function

1. Link Events

```
<body>
<div>
<a href="#" onClick="alert('Ooo, do it again!');">
Click on me!</a>
<a href="javascript:void('')" onClick="alert('Ooo, do it again!');">
Click on me! </a>
<a href="javascript:alert('Ooo, do it again!')" >
Click on me!</a>
</div>
</body>
```

Event Object (Events)

Note: No <script> tag.

- Anything that appears in the quotes of an onclick or an onmouseover is automatically interpreted as JavaScript.
- href="#" tells the browser to look for the anchor #, but there is no anchor "#", so the browser goes to the top of the page since it could not find the anchor.
- <a href="javascript:void(' ')" tells the browser not to go anywhere
  - it nullifies the link when you click on it.
  - href="javascript: is the way to call a function when a link is clicked.

Event Object (Events)

2. Actions within forms

```
<script>
function checkField(fld){
  if (fld.checkValidity() && fld.value>100) {
    alert("Correct number");
    fld.style.backgroundColor="white";
  } else {
    alert("Please enter a number greater than 100");
    fld.value=null;
    fld.style.backgroundColor="red";
    setTimeout(function() {
      document.getElementById('idname').focus();
      console.log("focused");
    }, 3000);
  }
}
function setStyle(fld) {
  fld.style.backgroundColor="yellow";
}
</script>
```

Event Object (Events)

```
<body>
<form id="frm1" action="form_action.asp">
<p>Insert a Number: <input type="input"
                           required pattern="^[0-9]+$" id="idname"
                           onfocus="setStyle(this)"
                           onchange="checkField(this)">
</p>
</form>
</body>
```

Event Object (Events)

3. Body onLoad and unLoad
    - These Event Handlers are defined in the <body> tag of an HTML file and are invoked when the document is fully loaded or unloaded.
    - If you set a flag within the onLoad Event Handler, other Event Handlers can test this flag to see if they can safely run, with the knowledge that the document is fully loaded and all objects are defined.
    - Used to check the visitor's browser type and version, and load the proper version of the web page
    - Used to deal with cookies that should be set when a user enters or leaves a page
    - onload - allows launching a particular JavaScript function upon the completion of initially loading the document
    - onunload - is triggered when the user "unloads" or exits the document

Event Object (Example)

```
<head>
<meta charset="utf-8">
<title> Event handler </title>
<script>
let loaded = "false";
function doit() {
  alert('Everything is "loaded" and loaded = ' + loaded);
}
function bye(){alert("Bye bye");} //Ineffective

</script>
</head>
```

Event Object (Example)

```
<body onload="loaded='true';" onunload="bye()">
<form id="frm1" action="form_action.asp">
<p>
<input type="button" value="Press me"
onClick="if (window.loaded) doit();">
</p>
</form>
</body>
```

Examples (keydown event)
Move a figure on the page

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title> Move a figure </title>
</head>
<body onload="set()">
<div id="mydiv" style="position:absolute; left:450px; top:300px;">
<img src="tower.jpg" alt="Tower">
</div>
<script src="./myscript12.js">
</script>
</body>
</html>
```

Examples (keydown event)
Move a figure on the page

```
// myscript12.js
var elem = document.getElementById('mydiv');
function set() {
  document.onkeydown = onKeyHandler;
}

function onKeyHandler(e) {
  e = (!e) ? window.event : e; //Explorer -> !e
  var key =(e.which != null) ? e.which : e.keyCode; //Firefox -> e.which
  switch (key){
    case 37: move(-3, 0); break; // left
    case 38: move(0, -3); break; // up
    case 39: move(3, 0); break; // right
    case 40: move(0, 3); break; // down
    default: window.alert('stop'); }
}

function move(dx,dy) {
  elem.style.left = parseInt(elem.style.left) + dx + "px";
  elem.style.top = parseInt(elem.style.top) + dy + "px";
}
```

Same example, recent API

- Some properties deprecated and replaced by new ones

```
function onKeyHandler(e) {
 switch(e.code) {
  case "KeyS": case "ArrowDown":
   move(0, 3);
   break;
  case "KeyW": case "ArrowUp":
   move(0, -3);
   break;
  case "KeyA": case "ArrowLeft":
   move(-3, 0);
   break;
  case "KeyD": case "ArrowRight":
   move(3, 0);
   break;
  default:
   window.alert('stop');
 }
}
```

KeyboardEvent.keyCode

Deprecated: This feature is no longer recommended. Though some browsers might still support it, it may have already been removed from the relevant web standards, may be in the process of being dropped, or

KeyboardEvent.code

The `KeyboardEvent.code` property represents a physical key on the keyboard (as opposed to the character generated by pressing the key). In other words, this property returns a value that isn't altered by keyboard layout or the state of the modifier keys.

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
code	48	79	38	No	35	10.1	48	48	38	35	10.3	5.0

Full support No support

images @ MDN contributors

Timer Events

setInterval(function, delay)

Calls a function repeatedly, with a fixed time delay between each call to that function. The setInterval() method will continue calling the function until clearInterval() is called, or the window is closed.

delay is expressed in millisec.

setTimeout(function, delay)

Calls a function after a specified number “delay” of milliseconds.

clearInterval(id_of_setinterval)

Clears a timer set with the setInterval() method. The ID value returned by setInterval() is used as the parameter for the clearInterval() method.

clearTimeout(id_of_settimeout)

Clears a timer set with the setTimeout() method. The ID value returned by setTimeout() is used as the parameter for the clearTimeout() method.

Timer Events

```
<head>
<meta charset="utf-8">
<title>setInterval/clearInterval example</title>
<script>
var intervalID;
function changeColor() {
  intervalID = setInterval(flashText, 1000);
}
function flashText() {
  var elem = document.getElementById("my_box");
  if (elem.style.color == 'red') {
    elem.style.color = 'blue';
  } else {
    elem.style.color = 'red';
  }
}
function stopTextColor() {
  clearInterval(intervalID);
}
</script>
</head>
```

Timer Events

```
<body onload="changeColor();">
<div id="my_box">
<p>Hello World</p>
<button onclick="stopTextColor();">Stop</button>
</div>
</body>
```

Examples

Dynamic Effects

- Dynamic effects are obtained by combining events and properties of the different objects
- Examples
  - Rollover – change of the images when the user passes over the images with the cursor of the mouse
  - Brief Animations
  - Sliding text
  - Gradual change of the background colour
  - Layer management

Rollover

```
<body>
<div>

<img src="a.jpg" height="100" width="100"
alt="a" onmouseover="modify(0,'over')"
onmouseout="modify(0,'out')">

<img src="b.gif" height="100" width="100"
alt="b" onmouseover="modify(1,'over')"
onmouseout="modify(1,'out')">

</div>
<script src="./myscript13.js">
</script>
</body>
```

Rollover

```
// myscript13.js
const img1 = new Array(2);
img1[0] = new Image();
img1[1] = new Image();
const img2 = new Array(2);
img2[0]= new Image();
img2[1]= new Image();
img1[0].src="a.jpg";
img1[1].src="b.gif"; //default images
img2[0].src="b.gif";
img2[1].src="a.jpg"; //rollover images

function modify(i,type) {
  if (type == "over")
    document.images[i].src = img2[i].src; //mouseover
  else
    document.images[i].src = img1[i].src; //mouseout
}
```

Sliding Text

```
<head>
<meta charset="utf-8">
<title> Sliding text </title>
<style type="text/css">
#slidText { color : red; background:black}
</style>
</head>

<body onload="setInterval(slide,100);">
<form action='#'>
<p>
<input type="text"
size="30"
name="mytext"
id = "slidText"
style="border: solid;">
</p>
</form><script
src="./myscript14.js">
</script>
</body>
```

Sliding Text

```
// myscript14.js
let text = "Example of Sliding Text .....";
function slide() {
  const firstchar = text.charAt(0);
  text = text.slice(1, text.length) + firstchar;
  document.forms[0].mytext.value = text;
}
```

Managing Layers

- Layers allow designers to (partially) superimpose an content on another content
- The element HTML div permits managing the layers using the following properties;
  - position: identifies the position of the element in the page
    - Values: absolute, fixed and relative
  - left, right, top, bottom: identify the position of the left-top or the right-bottom margin with respect to the reference container.
  - height and width: specify the width and height of the layer in pixels
  - z-index: specifies the layer (positive integer) on the z-axis
  - visibility: determines whether the element is visible or not.
    - Values: hidden and visible

Managing Layers Example

```
<head>
<meta charset="utf-8">
<title>
Properties of the levels
</title>
<link rel="stylesheet"
href="mystyleJS.css"
type="text/css"
media="screen">
</head>
```

Managing Layers Example

```
<body>
<h1>
Properties of the levels
</h1>
<div id="div1"
style=
"position: absolute; left: 300px; top:150px; width:300px; height:150px; z-index: 1">
Red
</div>
<div id="div2"
style=
"position: absolute; left: 350px; top:200px; width:300px; height:150px; z-index: 2">
Blu
</div>
<div id="div3"
style=
"position: absolute; left: 400px; top:250px; width:300px; height:150px; z-index: 3">
Green
</div>
```

```
<div id="control">
<form action="#" name="mymodule" id="mymodule">
<p>
1) Insert the block:<br>
<select name="idiv">
<option value="div1" selected> red </option>
<option value="div2"> blu </option>
<option value="div3"> green </option>
</select><br>
2) Insert the distance from the top:<br>
<input type="text" size="5" name="top" value="150">
<br>
3) Insert the distance from the left:<br>
...
<script type="text/javascript"
src="./myscript15.js">
...
```

Managing Layers
Example (myscript15.js)

```
// myscript15.js
var ERROR = false;
function read(i) {
  value = 0;
  with (document.mymodule) {
    switch (i) {
      case 1: return idiv.value;
      case 2: value = top.value; break;
      case 3: value = left.value; break;
      case 4: value = width.value; break;
      case 5: value = height.value; break;
      case 6: value = level.value;
    }
  }
  value = parseInt(value);
  if ((value < 0) || isNaN(value)) {
    window.alert("Error in the field n." + i);
    ERROR = true;
  }
  return value;
}
```

Managing Layers
Example (myscript15.js)

```
function set() {
  var i = read(1);
  var x = read(2);
  var y = read(3);
  var w = read(4);
  var h = read(5);
  var z = read(6);
  if (!ERROR) {
    var mydiv = document.getElementById(i);
    mydiv.style.top = x + "px";
    mydiv.style.left = y + "px";
    mydiv.style.width = w + "px";
    mydiv.style.height = h + "px";
    mydiv.style.zIndex = z + "";
   //(1)
  }
  ERROR = false;
}
```

Managing Layers
Example (mystyleJS.js)

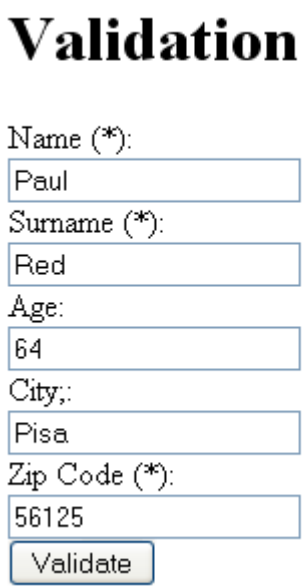
```
div {
  border: outset;
  font-size: xx-large;
  font-weight: bolder;
  overflow: hidden;
}

#div1 { background-color: red;}
#div2 { background-color: blue; }
#div3 { background-color: green; }

#control {
  background-color: azurine; z-index: 0;
  width: 15em; padding: 5px;
  font-size: medium; font-weight: normal;
}
```

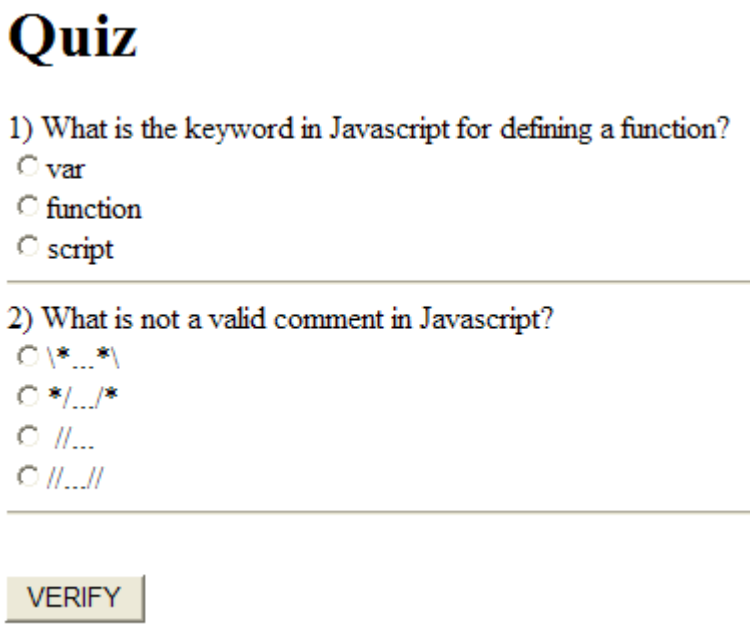
Validation of data inserted into a Form

- JavaScript allows verifying the data inserted into a form directly on the client-side
- Using handlers of appropriate events (onclick and onblur), the data can be analysed and messages can be sent to the user
  - onclick – when the form is completed and sent to the server
  - onblur – when the focus is passed on another element
- Typically, the verification is performed only on the input text fields

Validation of data inserted into a Form

- Two examples
  - Validation of data
  - Assessment of the answers to a quiz





Validation of data inserted into a Form

```
let fields = document.getElementsByTagName("input");
const shouldBeALPHANUMERICAL = [0, 1, 4];
const shouldBeNUMERICAL = [2, 4];
const shouldBeALPHABETICAL = [0, 1, 3];
const shouldBe5NUMBERS = [4];
const MESSAGES = ["The following field does not have valid symbols: ",
"The following field is not exclusively numerical: ",
"The following field must have five digits: ",
"The following field must have only letters: ",
"...",
"OK"];

const existALPHANUMERICAL = /\w/;
const existNONALPHANUMERICAL = /\W/;
const existNONNUMERICAL = /\D/;
const existNUMERICAL = /\d/;
const exist5NUMERICAL = /\d{5}/;
```

Validation of data inserted into a Form

```
function error(idmess, field) {
  window.alert(MESSAGES[idmess] + field.name);
  field.focus();
  field.select();
}
function isTrue(cond, elem, mustBe, mess) {
  for (var i = 0; i < elem.length; i++) {
    let j = elem[i];
    let field = fields[j];
    if (cond.test(field.value) == mustBe) {
      error(mess, field);
      return true;
    }
  }
  return false;
}
```

Validation of data inserted into a Form

```
function validate() {
  if (isTrue(existALPHANUMERICAL, shouldBeALPHANUMERICAL, false, 0)
    return;
  if (isTrue(existNONALPHANUMERICAL, shouldBeALPHANUMERICAL, true, 0))
    return;
  if (isTrue(existNONNUMERICAL, shouldBeNUMERICAL, true, 1))
    return;
  if (isTrue(exist5NUMERICAL, shouldBe5NUMBERS, false, 2))
    return;
  if (isTrue(existNUMERICAL, shouldBeALPHABETICAL, true, 3))
    return;
  if (isTrue(existNONALPHANUMERICAL, shouldBeALPHABETICAL, true, 3))
    return;
  window.alert(MESSAGES[MESSAGES.length - 1]);
}
```

Assessment of the answers to a quiz

```
<form action="#" name="mymodule" id="mymodule">
<script>
for (let i = 0; i < Queries.length; i++) {
  document.writeln("<div>" + (i + 1) + ") " + Queries[i] + "<br>");
  for (let j = 0; j < Options[i].length; j++)
    document.writeln("<input type='radio' name='q" + i + "'>" +
  Options[i][j] + "<br>");
  document.writeln("<hr></div>");
}
</script>
<p>
<input type="button" value="VERIFY" onclick="check()">
</p>
</form>
```

Assessment of the answers to a quiz

```
//myscript18.js
const Queries = ["What is the keyword in Javascript for defining a function?",
"What is not a valid comment in Javascript?",
"What is the handler for the 'loss of active state' event?",
"Which operator can be used to instance an object?",
"To periodically call a function you use the method:"];

const Options = [["var", "function", "script"],
["\\*...*\\", "*/.../*", " //...", "//...//"],
["onFocus", "onBlur", "onClick"],
["create", "new", "add"],
["window.setInterval", "window.setTimeout", "date.setTime"]];

const Answers = [0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0];
```

Assessment of the answers to a quiz

```
function check() {
  let score = 0;
  let answers = document.getElementsByTagName("input");
  for (var i = 0; i < answers.length - 1; i++)
    if (answers[i].checked)
      if (Answers[i] == 1) score++;
      else score--;
  window.alert("Your score is: " + score);
  document.mymodule.reset();
}
```