User interactivity

MGR. DANIELA PONCE, PHD.

(2019)

Lecture content

- Input elements
- Document object model, browser object model
- Application interface
- JavaScript elements (variable, command, attribute, method, object)
- Expressions, commands, functions, dot notation
- Events
- Users without JavaScript

Input elements

- HTML element, in which the user can enter values, select from possible values or start an action
- Enable the page interactivity with user
- Input data are available at the side of:
  - client: the script inserted/linked to the document (mostly JavaScript program)
  - server: program running at the server (e.g. PHP application or again JavaScript) will process the data after their submission

HTML input element: input

- Input field
- Impair HTML5 element, no content, attributes only (type and others)
- Several variants, according to the values of the type attribute

Example:

- <input type="text" id=„name“ value=„Your name">

Input variants according to the value of the type attribute

- text – one row input text field
- number – input number field
- password – password field (entered characters are masked)
- checkbox – checkbox field
- radio – switch field
- select – drop-down list, with options
- option – items at drop-down list

Example:

```
<select id="greeting">
    <option value="mr">Mr.</option>
    <option value ="ms">Ms.</option>
</select>
```

- More possibilities:
  - color, file, search, image, hidden, date , datetime-local, email, month, range, tel, time, url, week (details)

Buttons

- A kind of input element
- Impair element <input>, setting of the type attribute:
  - button – common button linked to a function
  - reset – reset button (clear all elements in the form)
  - submit – submission button (will send form data to server)
  - Suitable for forms
- Pair element <button>:
  - The type attribute can be set to button, reset, submit values
  - A combination of text and image can be used in its label

- Example: <input type="submit" value=„Create an account" >
- Example: <button type="button">Click on me!</button>

More attributes of the <input> element

- value .. (default) value of the element
- checked .. Checked option (for switch and checkbox fields)
- autocomplete .. on/off .. Autocompletion based on previously entered values
- placeholder .. A short hint describing the value expected
- maxlength .. max. number of characters that can be entered
- min, max .. min, max. possible value/date
- size .. Size of the element in characters
- list .. List of predefined possible values
- pattern .. Regular expression for the control of the value entered

The <label> element

- Description of the input element
- Informs the user what kind of value should be entered
- If interconnected with the input element, behaves as a part of the input element (e.g. clicking on the label will place the cursor into the input field)
- How to interconnect: identical value of the for (label‘s) and id (input‘s) attributes

Example:

```
<label for=„anElement">Name:</label>
<input type="text" name=„elementName" id="anElement">
```

Javascript – programming language

- Created in 1995 (Brendan Eich), ECMA standard (1997)
- Official name of the standard: ECMA-262
- Official name of the language: ECMAScript
- Object-oriented (object, method, interface, encapsulation, …)
- Multiplatform (works at every platform / operational system)
- Interpreted
  - The source code in JS is step by step translated and executed by the browser
- Inserted / attached to html pages
- Functional code executed inside the browser at the client side

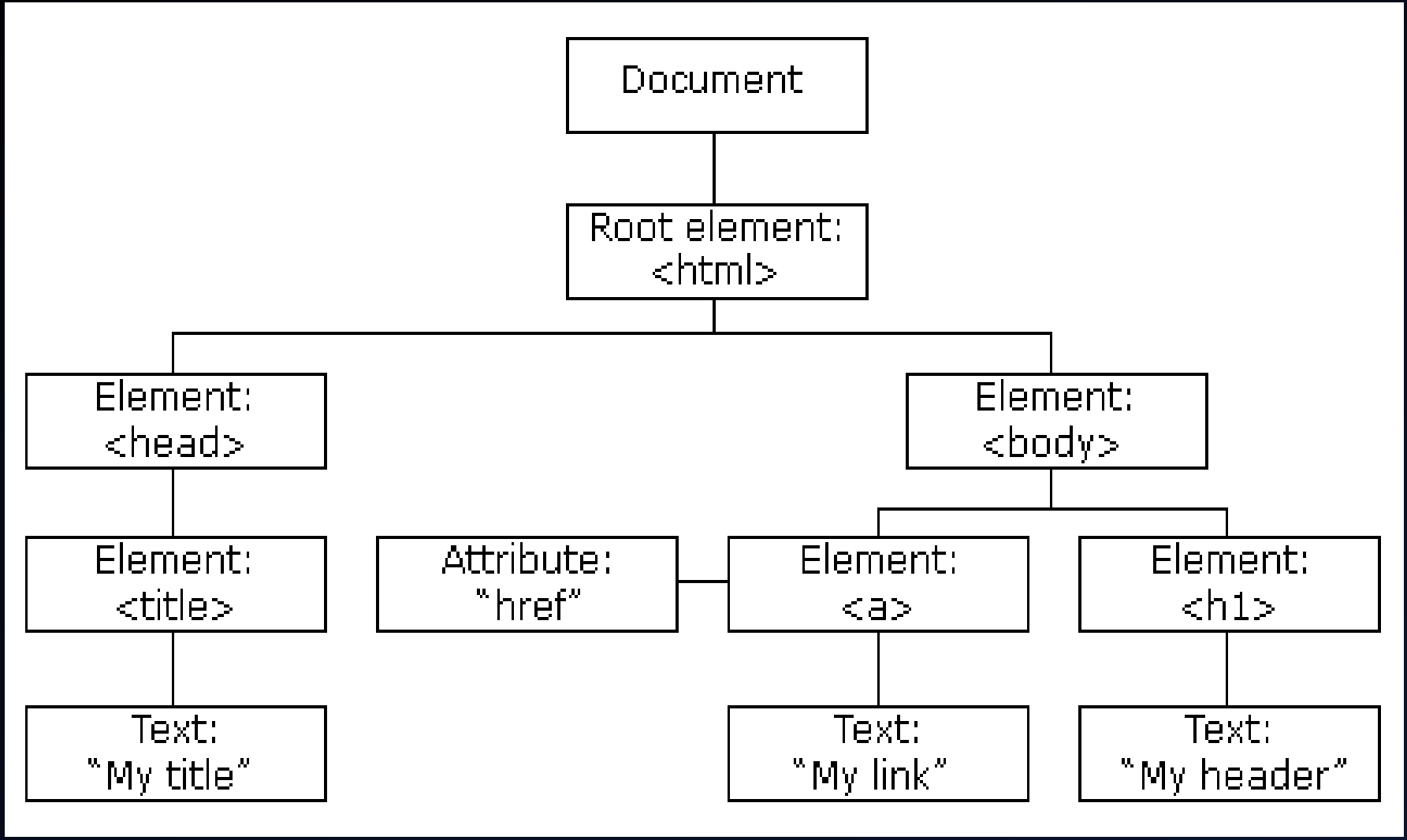
Possibilities of use

- Change of the HTML content
  - document.getElementById('myImage').src='pic_bulbon.gif';
  - document.getElementById('myImage').src='pic_bulboff.gif';
  - document.getElementById("demo").innerHTML = "Hello JavaScript";
- Change of the cascading styles / classes assigned to elements
  - document.getElementById("demo").style.fontSize = "35px";
  - document.getElementById("demo").style.display = "none";
  - document.getElementById("demo").style.display = "block";
- Reaction to events on the page
  - button.onclick = function(){ … }
- Validation of the user input (forms)

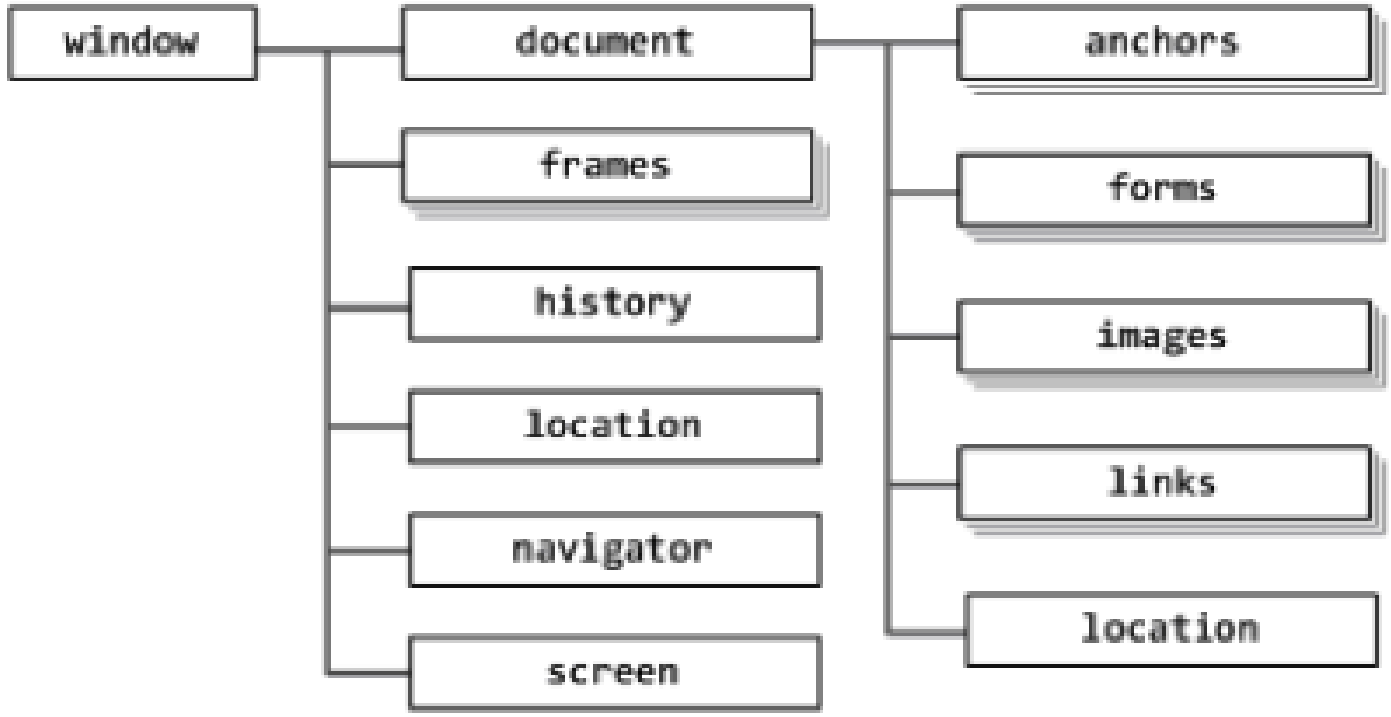
DOM, BOM, API

- **D**ocument **O**bject **M**odel, **B**rowser **O**bject **M**odel, Application Programming Interface
- Object oriented representation of the HTML document (DOM) and browser (BOM), resp., in the form of tree structure
- Collection of classes (their methods), functions and interfaces of a library which can be used by another program
- DOM API
  - Access to the content, content change or structure change of the document and its parts for programs, e.g. in JavaScript
- BOM API
  - Access to the address bar, browser history and its functions (e.g. refreshing the content of the browser window)
  - Access to the elements and operations of the browser for other programs, e.g. in JavaScript

Document Object Model (example)



BOM (browser object model)



Tree structure of the object model DOM/BOM

- Node: every element in the DOM/BOM tree.
- Root node: node with descendants only
- Direct descendant: a node directly subordinated to some node (connected by an edge)
  - Title is a direct descendant of the head node
- Descendant: node subordinated to some node, not necessarily by an edge
  - Title is a descendant of the html node
- Parent node: directly superordinated to some node (it is a node with descendants)
- Sibling nodes: nodes with the same parent
- Text node: node containing text (node without descendants)

DOM and BOM relation

- BOM (Browser Object Model):
  - Root node: window
  - Direct descendatnts: navigator, history, screen, location and document objects
  - No standard (different browsers may have different BOMs)
- DOM (Document Object Model) : document object, which has further parts:
  - Root node: html
  - Direct descendants: head, body
  - Standard structure for all browsers (different browsers have the same DOM structure)

JavaScript basic element

- Variables:
  - Temporal storing of plain (unstructured) information
  - Enable later processing of this information
- Example: variable *name*, operations *compareNames*, *findPerson*
- Objects:
  - Structure for storing of related information
  - Properties – items for storing parts of the complex information
  - Methods – procedures for processing properties and input data
- Example:
  - Object *joeDoe*, attributes *name* and *surname*, method *writeCompleteName*

Variables declaration and dot notation

```
const h1 = document.querySelector('h1');
let name = localStorage.getItem(name');
h1.textContent = Hello, ‘ + name + ‘!‘;
```

Explanation:

- Declaration of constants (will not change): const
- Declaration of variables (will change): let (or var)
- Access to the method of the object (method calling):
  - document.querySelector(), localStorage.getItem()
- Access to the attribute (here value assignment):
  - h1.textContent = …

Variables

- Must be first declared, then used (let or var keyword)
- JS is CASE sensitive
- Cannot be used in names: – hyphen
- Can be used in names: _ underscore
- Nontyped variables: when declaring, the type of the variable is not known (number, string, boolean, …)

```
<script>
  let number = 1;              // LET or Let cannot be used for declaration
  let text = “hello“;
  let length = text.length;    // the length variable is 5
</script>
```

Commands and operators

- Commands are separated by semicolon ; (or by end of the line)
- = value assignment to the variable
- + summing numeric values:
- var a = 5+5; var b=5; var c = a+b; alert(c); // vypíše 15
- + concatening two character strings:
- var a = 5+5; var b=„5“; var c = a+b; alert(c); // vypíše 105
- Value comparing:
  - == equals 5 == "5" // ture
  - === equals and has the same type 5 === "5" // false
  - != does not equal 5 != "5" // false
  - !== does not equal or its type is different 5 !== "5" // true

Logic operators, conditioned commands

- Logic operators:

  && logic and

  || logic or

  ! negation

- Logically conditioned commands:
  - Execution is depending on the true value of the logic condition
  - Can be nested
- Syntax:

  if (*logic_expression*) { *then_action_sequence* }

  else { *else_action_sequence* }

Example (conditioned commands)

```
if (hour < 12) { greeting = “Good morning!";
                                    // when the condition is true
} else { greeting = „Good afternoon!“; }
                                    // when the condition is false
```

Function

- Declarated (and optionally named) part of the code, which processes data (and optional input parametrs), (optionally returns output value) and is called from some other part of the code

```
<script >
//....
function removeItem () {
    list.removeChild(listItem);
    totalCost = Number(totalCost) - Number(myCost);
    summary.textContent = ,Total cost: , +totalCost +',-Kč.';
}
listButton.onclick = removeItem;
< /script>
```

JS program output

- Writing to the innerHTML attribute of an HTML element, the rest of the document without any changes
  -
    <script>
    document.getElementById("demo").innerHTML = 5 + 6;
    </script>
- Writing to the HTML document, function document.write()
  - Debugging only; document will be completely overwritten(!)
  - <script>document.write(5 + 6);</script>
- Writing to the pop-up alert window, function window.alert()
  - <script>window.alert(5 + 6);</script>
- Writing to the console of the browser, function console.log()
  - <script>console.log(5 + 6);</script>

Objects (summary)

- Object is a colletion of named values (attributes)
- Composed of attribute name and attribute value pairs
- Object has methods for work with attributes
- Method is a property of the object, and it contains definition of a function (code) performed on the object

Object (example)

- Creating link object (object of HTML *a* type):
  - let myLink = document.querySelector('a'); // first link in DOM will be returned
- Properties textContent, href, …
- Assigning values to attributes:
  - myLink.textContent = 'Mozilla Developer Network';
  - myLink.href = 'https://developer.mozilla.org';

- Method calling:
  - myLink.setAttribute('class', ,accent'); // setting the property *class* for myLink object

Accessing HTML objects

- Objects in the entire HTML document:
  - element = document.querySelector(selectors)
    - First element satisfying the expression in the brackets will be returned
    - The expression in the bracktes has to be valid in CSS language (can be composed CSS expression)
  - elementList = document.querySelectorAll(selectors);
    - All elements satisfying the expression in the brackets will be returned
    - Details on the querySelectorAll method

- Older methods:
  - getElementById(id)
  - getElementsByTagName(tagName)

Event (JS constructs event and event listener)

- In HTML5, it is possible to react to events by JS program
- Events can be related to, e.g. :
  - Document (loading, printing, refreshing, loosing connection)
  - Browser window (window focus, history, hiding/displaying, size changes)
  - Mouse (clicking, double clicking)
  - Cursor (pointing by the cursor to an element)
  - Form (form submitting)
- Event can be followed by a function(s) (named event listener), which have to be executed ahen the event occurs
- The same event listener can be attached more than one events
- Attaching event and event listener can be declared in both HTML and JS

Events - example

```
…
<label for="item">Give new item:</label>
<input type="text" name="item" id="item">
<button>Add</button>

<script>
let input = document.document.querySelector('item');
let button = document.querySelector('button');
function addItem() { …
    input.focus();
}
button.addEventListener(“click“, addItem);
< /script>
…
```

Events in HTML and in JS

- We have previously defined a JS function myAction:
  - function myAction () { … }
- We wish to attach it to the click event on a button

1. Event types as JS constructs (objects implementing the Event interface):
   - afterprint, beforeprint, click, doubleclick, load, online, focus, blur, reset, submit, …
   - Complete list: https://developer.mozilla.org/... /Events
- Solution in JS (preferred): myElement.addEventListener("click", myAction)

2. Event types as attributes of HTML element:
   - onafterprint, onbeforeprint, onclick, ondblclick, onload, ononline, onfocus, onblur, onreset, onsubmit, …
   - Complete list: https://www.w3schools.com/.../ref_eventattributes.asp
- Solution in HTML: <input type="button" onclick="myAction()">

Example:
Attaching functions to document loading event (3 options)

1. document.body.addEventListener(“load“, initialization);
   - In JS, we call addEventListener method of the document.body object, which will register the initialization function for the load event
   - Later on, more functions can be registered for the same event
   - Preferred way
2. document.body.onload = initialization~~()~~;
   - In JS, we set the onload attribute of the document.body objectu to the *definition* of the initialization function (therefore no brackets)
   - Later no, more functions can NOT be registered for the same event
3. <body onload=“initialization()“>
   - In HTML, we assign the initialization function to the onload attribute of the body element

Placing JavaScript in HTML document

- JavaScript code has to be placed in between <script> and </script> HTML marks
- Document may contain arbitrary number of <script> elements
- Scripts can be placed in the body as well as in the head of the HTML document (<body>, <head> elements)
- If the script is placed at the end of the body, it will be only compiled after the complete page is loaded and thus the page rendering becomes faster (code compiling makes page rendering slower)

Placing JavaScript code outside HTML document

- JavaScript in a separate text file with .js extension
- Using the parametr defer the script execution can be postponed untill the complete page gets loaded
- The name of the file with scripts is declared in the <script> element:
  - <script src="myScripts.js" defer></script>

    Several script elements can be used, too:
  - <script src="myScript1.js" defer></script>
    <script src="myScript2.js" defer></script>
- Advantages of placing the JS in external file:
  - HTML content and JS code (behavior) separation
  - Easier understanding and maintenance of the content and behavior
  - Once loaded JS files in cache memory make page loading faster

Document in browser without JavaScript

- User can forbid JavaScript
- Browser does not necessarily support JavaScript
- Solution: to use <noscript> HTML element
- Can be placed in the <head> or <body> elements of the document
  - If placed in the <head> element, it may only contain <link>, <style> and <meta> elements
- The content of the noscript element will be processed if only JavaScript is not supported by the browser

Example: using the noscript element

- It is convenient to hide JS code in HTML comment
- If the browser can not interpret the script element, it will be ignored and the JS code itself will become HTML comment (and it will not be displayed at the screen by the browser)

```
<noscript>
 The browser does not support JavaScript.
 </noscript>
```

```
<script>
<!-- hide JS code; will be skipped by JS
function displayMsg() {
    alert("Hello World!")
}
// comment the end of the HTML
comment in JS -->
</script>
```

Know the browser of your user

- It is not necessary to think of all kinds of device, all kinds of browsers, all versions of browsers
- Who is the expected user of our web site?
  - General data e.g. at GlobalStats (by browsers, versions, screen resolution, device type, region, …)
- Who is the real user of our web site?
  - Track your users and then optimize

Resources

- Live DOM:
  https://software.hixie.ch/utilities/js/live-dom-viewer/
- JavaScript Guide, tutorial:
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
  https://developer.mozilla.org/en-US/docs/Learn/JavaScript
  http://www.w3schools.com/js/default.asp
- Browsers compatibility, JS :
  https://quirksmode.org/compatibility.html
  http://kangax.github.io/compat-table/es6/