Faculty of Science,
Technology
and Medicine

Web Programming

Volker Müller
University of Luxembourg

JavaScript

Scripting language, dynamic, weakly typed, prototype-based

Traditionally used for client-side web development

Started by Netscape in 1995, in 1996 MS followed with JScript

Implementation of ECMAScript standard

More Information and Tutorials

Specification ECMAScript, 11th edition (2020):

http://www.ecma-international.org/ecma-262/11.0

Tutorial and many examples:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

http://www.w3schools.com/JS

http://openbook.rheinwerk-verlag.de/javascript_ajax/

JavaScript Security

JS runs in a "sandbox" inside browser

Can originally only access browser objects, not access file system or local files (changing a lot with recent versions)

Some manipulations of browser objects need user agreement

Same origin policy: JS Code within sandbox is generally restricted from "interacting" resources from other origins (scheme/host/port)

Cross-Origin Network Access

Cross-origin writes / embedding typically allowed

Cross-origin reads are normally disallowed

CORS (Cross-Origin Resource Sharing) uses HTTP headers ("Access-Control-Allow-Origin") to give apps access to resources from different domain

See developer.mozilla.org for a detailed description of the rules

Usage Example (1)

Code directly embedded in HTML (in head or body):

```
<script type = "text/javascript">
    alert ("Hello World");
</script>

<button onclick="myFunction()">Click</button>
```

Usage Example (2)

Link to external file (typically in header):

```
<script type = "text/javascript" src = "hello.js">

</script>
```

Code in hello.js from same origin as html page

File hello.js contains JS code (without any surrounding tags)

Comments on Usage

Several script blocks possible, will be concatenated

Can be in header or body:

- Often: function definitions loaded in header
- Directly runnable code / function calls in body

Code position might have large impact on rendering performance

Basic Features

Variables: var x = "Volker", y = 1;

let z = 3;

"let" block scoped, "var" variables in global context

Standard operators (almost equal to PHP)

Conditionals: if (...) { ... } else { ... }

Loops: while-loop, do-while loop, for-loop, break, continue

Strings in JS

No difference between strings in single or double quotes, but also string object "String" exists

Template literals / template strings are string literals enclosed by backtick (` `), allowing embedded expressions, multi-line strings and string interpolation features:

```
console.log(`Fifteen is ${a + b} and

not ${2 * a + b}.`);
```

Functions

```
function sqr (x)
{
  let y = x*x;
  return y;
}

 ....

var x = sqr (5);
```

"let" is optional, but if not used, then assignment to variable "y" will overwrite possibly existing outside variable "y" (like "var")

Some Predefined Functions

eval (string) → deprecated, do no longer use

parseFloat (string), parseInt (string)

isNaN (string), isFinite (int)

Number (expression), String (expression)

escape (string), unescape (string)

Closures

Unnamed function with free variables that are bound in the lexical environment

```
function bestSellingBooks(threshold) {

  return bookList.filter(

      function (book)

        { return book.sales >= threshold; }   );

}
```

Arrow Functions

Arrow function = shorter syntax for anonymous function expressions:

```
var e = ["a", "ab", "abc"]

e.map(element => element.length)
```

equivalent to

```
e.map(function(element) { return element.length; })
```

Similar for more arguments: `(a, b) => a + b`

New since JS 8th edition: Promises

JS is single threaded → two bits of script cannot run at same time → can create delays during waits

A Promise is an object encapsulating a function that can only succeed or fail once

Extremely useful for async success/failure

https://developers.google.com/web/fundamentals/primers/promises for a detailed description

General Idea of Promises

Calling a synchronous function → return value shows success / failure (or there is thrown exception)

Code following function call handles success / failure situation

Calling an asynchronous function → return value not yet available for code following function call

–> Asynchronous function code encapsulated in Promise object, "linked" with success & error handler when called

Promise Example

```
var p = new Promise(function(resolve, reject) {
  // do something asynchronously
  if (/* everything turned out fine */) {  resolve("Worked!"); }
  else { reject("Failed"); }     });
```

Using the promise object

```
p.then(
 function(result) { console.log("OK " + result); },
 function(err) {  console.log("ERROR " + err); }
).catch( /* function for catching exceptions */ )
```

New in JS 8th edition: async functions

Allow you to write promise-based code as if it were synchronous, but without blocking main thread

```
async function logFetch(url) {
  try {  const response = await fetch(url);
        console.log(await response.text());
  }
  catch (err) { console.log('fetch failed', err);  } }
```

See more info at https://developers.google.com/web/fundamentals/primers/async-functions

Objects in JavaScript

```
var person = {
  name: ['Bob', 'Smith'],
  age: 32,
  bio: function() {  alert (this.name[0]); }
....
}
```

Object = instance of a class

Objects (2)

Attributes have no visibility, just access them directly, can be assigned other objects:

```
person.age = 60;
```

Member variables / functions can be defined separately and included into object at any time through assignment (so objects can change at runtime)

Usage of member functions as in Java:

```
person.bio ( );
```

Sub-Objects

```
var person = {
 name : {
     first: 'Bob',
     last: 'Smith'
   }, ....
```

Now property name holds sub-object

person.name.first accesses sub-property

Adding Members & Constructor Functions

Member variables or functions can be added at any time:

```
person.farewell = function() { alert("Bye everybody!"); }

person.eyecolor = "brown";
```

Constructor function creates class Person

```
function Person(name) {
  this.name = name;
  this.greeting = function() {
    alert('Hi! I\'m ' + this.name + '.');
  };
```

Now object "Person" exists, instantiated with new Person("Volker")

Classes (since ECMAScript 2015)

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
 calcArea() {  return this.height * this.width;   }
}
```

Class Features

constructor is special method used to initialize class instance (only one constructor possible)

Static class methods exist (keyword "static")

Used with the class name instead the class instance (variable) name

Inheritance possible with "extends"

Prototype

We can add new properties to single object directly:

```
var s = new Rectangle();     s.color="blue";
```

Adding new properties to all instances (not overwriting property for objects where already existing):

```
Rectangle.prototype.color = "blue";
```

For-in Loop – Loop over Object attributes

```
function TextAttributes (obj)
{
  let attribute, result = "";
  for (attribute in obj)
   {
      result += String (attribute) + "<br/>";
   }
   ...
```

Predefined Object: Arrays

```
var a = new Array (2.3, 4.5, 7.2, 3.3);
```

Index starts always with 0

Predefined methods: length, concat (a), join (separator), pop( ), push (e1,e2,...), reverse ( ), shift( ), slice (start, stop), splice (start, count, e1, e2,...), sort ( ), sort (comparefunction), unshift(e1, e2,...)

Many more predefined objects exist

Exceptions

```
try
{  non_existant_object.x = 3;  }
catch (e)
  {   window.alert ("Error: " + e);   }
```

try – catch – final variant also exists

Throwing exceptions: `throw "Error";`

Browser Object Models

Browser object models allows JS to "talk to" the browser

Unfortunately, no standard exists

window: represents open window in browser

navigator (window.navigator): contains information about browser

screen (window.screen), history (window.history), location (window.location)

Some Attributes of Object "window"

- frames
- document
  - all
    - style
- anchors
- applets
- embeds
- forms
  - elements
    - options
- images
- layers
- links
- event

Many many more exist (see e.g. MDN site)

Example: Accessing Form Information

```
<form name = "demo">
<select name = "w" size="2"
    onChange = "chooseOption (  )">
<option>Option1</option>
<option>Option2</option>
</select>
```

chooseOption will be explained on next slide

Value of selectbox can be accessed as:

document.forms[0].elements[0] (if first form in doc.)

document.demo.w

Example: Form Information (2)

```
function chooseOption ( )
{
  var n = document.demo.w.selectedIndex;
  alert ("Choosen index : " + n);
  document.demo.textfield1.value = n + " --> " +
        document.demo.w.options[n].value) ;
}
```

Sets value of "textfield1" in form

Example: Changing Event Handlers

Most properties are even writable:

```
function init( )   // called as   <body onload="init()">
{
   document.onmouseup = setStatus
}
```

onmouseup defines default function called for "mouse up" event

Event now handled by user-defined function setStatus

Extension of this Idea

Find a possibility for JS Scripts to access all possible parts of an HTML file, not only forms, images, links, ...

Define how a document can be represented as an object, then we can apply the same ideas as before

Document Object Model (DOM)

Platform- and language-neutral interface

Allows programs to dynamically access and update the content, structure and style of documents

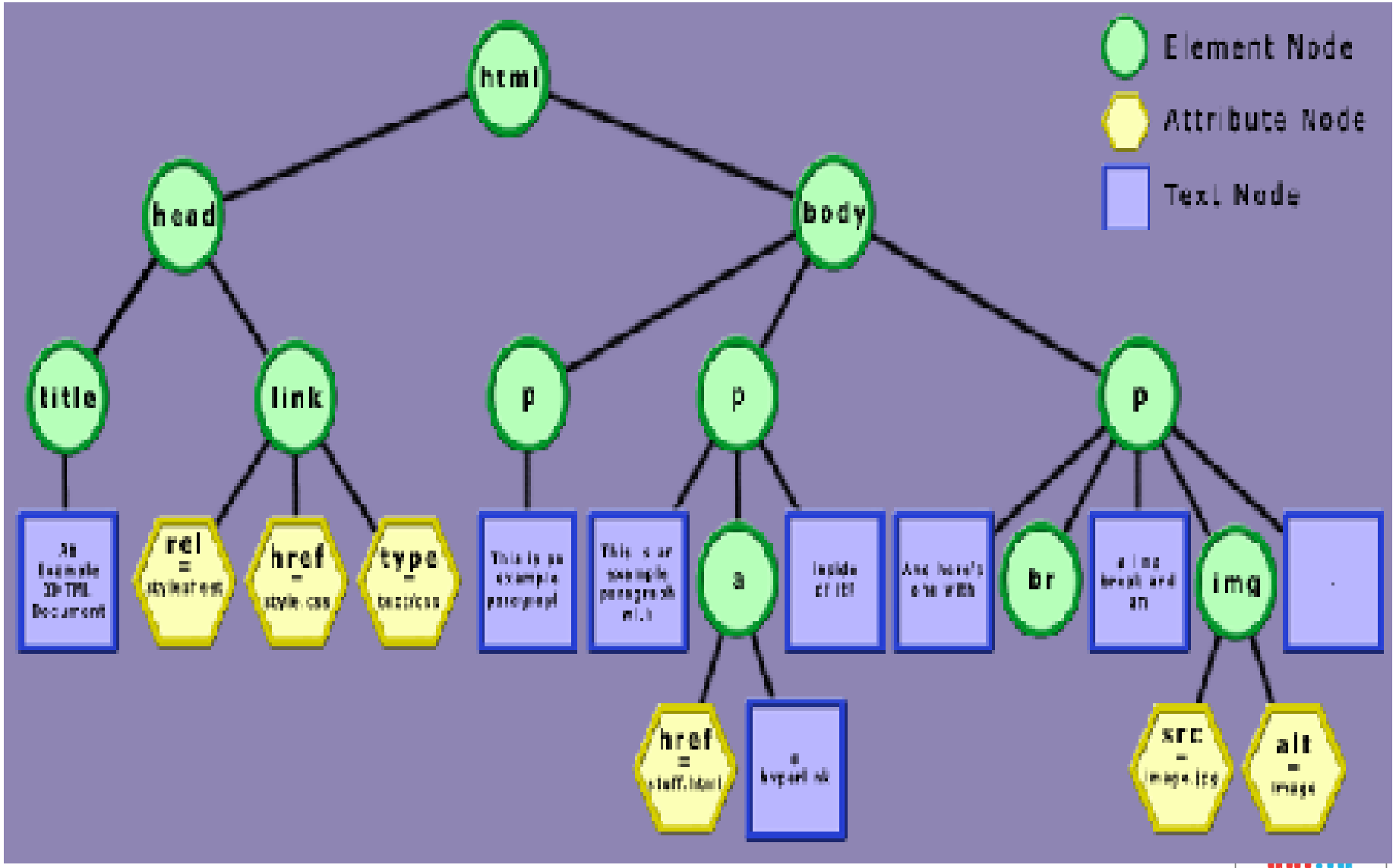
Focused on XML documents, but can also be used with HTML

Specification: http://w3.org/DOM

An Example HTML File

```
<html>

<head> <title> An Example HTML document</title>

<link ref = "stylesheet" href="style.css" type="text/css">

</head>

<body><p>This is an example paragraph.</p>

<p>This is an example paragraph with <a href="stuff.html">a
   hyperlink</a> inside it!</p> <p>And here's one with<br/>a line
   break and an <image src="image.jpg" alt="image">.</p>

</body></html>
```

DOM Tree for Example HTML



Main Idea of Using DOM

DOM specifies objects for representing XML documents, i.e. attributes and methods

Access of DOM subtrees possible

Object attributes can be changed

DOM tree can also be manipulated with methods:
change existing node, add new node, move nodes to another position, find parent node, ...

Example: DOM Interface for JS

```
interface HTMLDocument : Document {
  attribute DOMString       title;
  readonly attribute DOMString       referrer; ....
  readonly attribute HTMLCollection  images;
  readonly attribute HTMLCollection  forms; ...
  void  write (in DOMString text);
  void  writeln (in DOMString text);
  NodeList  getElementsByName (in DOMString elName);
```

Some Methods for Documents

Element createElement (in DOMString tagName)

Text createTextNode (in DOMString data)

Attr createAttribute (in DOMString name)

NodeList getElementsByTagName (in DOMString tagname)

Element getElementById (in DOMString elementId)

Example of Using DOM with JS

```
function addParagraph ( )
{
  var p = document.getElementById ("p1");
  var newP = document.createElement ("p");
  var newText = document.createTextNode ("New");
  newP.appendChild (newText);
  p.parentNode.appendChild(newP);
}
```

DOM Nodes

createElement ('element_name')

createTextNode ('text content')

Methods for nodes:

- attributes[ ]
- getAttribute('attributeName')
- removeAttribute('attributeName')
- setAttribute('attribute_name','attribute_value')

Attributes / Methods for Nodes & NodeList

parentNode

childNodes

firstChild, lastChild

previousSibling, nextSibling

getAttribute (name), setAttribute (name, value)

length

item(i)

See http://www.howtocreate.co.uk/tutorials/javascript/domstructure

Example: DOM Events

```
var oDiv = document.getElementById ('thediv');

oDiv.addEventListener('click', function (e) {

  alert('1st event handler'); });

oDiv.addEventListener('click', function (e) {

  alert('2nd event handler');});
```

Comments on DOM Event Handling

Can add several event handlers for the same event on same object, but order is undefined

There exists optional third boolean parameter that determines order of event handler calls (of same type) for inheritance chain of objects

Default ("bubbling" phase): event handlers called in up going inheritance order

DOM and Styles

DOM Level 2 defines access to CSS styles

Can also manipulate styles directly:

```
t = document.getElementById ("p1");

t.style.backgroundColor = "green";
```

Note: CSS property names slightly changed

background-color → backgroundColor

font-variant → fontVariant

JS DOM Problems

DOM API provides a nice framework for implementing very general "dynamic HTML"

Focus still on programmers, not always easy to develop scripts for “amateurs” → use simple JS Frameworks

Next Week

Ajax: Asynchronous Javascript and XML