JavaScript et DOM

Pour dynamiser des pages web !

Cécile HARDEBOLLE
cecile.hardebolle@supelec.fr

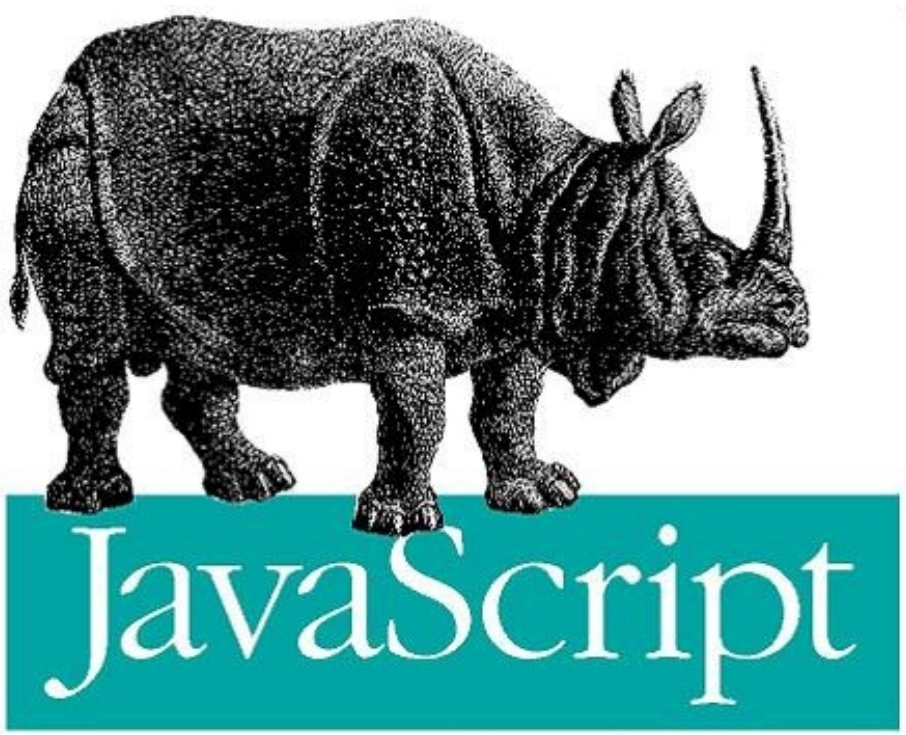
Prérequis

- Pratique d'un ou plusieurs langage(s) de programmation
  - Exemple : Java
- Compréhension de la notion de type
- Compréhension de la notion de fonction
- Compréhension et pratique basique de l'approche objet (classe, constructeur, attribut, méthode)

Objectifs

Être capable de :

- Ecrire un programme JavaScript mettant en œuvre des structures de contrôle, des fonctions et des objets
- Utiliser JavaScript pour traiter des événements liés à l'interaction de l'utilisateur avec une page web
- Utiliser JavaScript et DOM pour modifier le contenu d'une page web
- Expliquer les différences entre JavaScript et d'autres langages de programmation



Crédit : O'REILLY

Fin de quelques idées reçues…

- JavaScript ≠ Java !

- JavaScript = langage de programmation conçu pour écrire des scripts interprétés par un navigateur web … mais pas que !

- Principales caractéristiques :
  - impératif avec typage faible/implicite/dynamique
  - orienté objet « à prototype »
  - fonctionnel

Voir aussi « duck typing »…

Un peu d'histoire

- 1995 :
  - création de LiveScript par Brendan Eich pour Netscape Communication Corporation
  - Création d'une version côté client de LiveScript, baptisée JavaScript
- 1996 : interpréteur JavaScript pour Netscape Navigator 2.0
- 1997 : standardisation par l'ECMA sous le nom d'ECMAScript
  - Spécification : ECMA-262 (ES1)
  - Implémentations : JavaScript, ActionScript, JScript…
- 2005 : révolution AJAX !
- 2009 : ES5, "use strict", JSON…
- 2012 : ES6 en cours de définition (draft révision 20 le 28/10/2013)

Nouveautés prévues dans ES6

- Variables locales
- Classes et héritage
- Modules et import
- Fonctions
  - Valeur par défaut pour les paramètres
  - Nombre de paramètres indéfini
- Boucle for-of, iterator
- Map, Set, WeakMap
- …

Sources :

- The State of JavaScript, Brendan Eich / Strange Loop 2012
  http://brendaneich.github.com/Strange-Loop-2012/
- ECMAScript 6, Martin Catty, Synbioz
  http://www.synbioz.com/blog/ecmascript_6

Quelques références

- Pour tester du JavaScript :
  - L'ardoise de Firefox : https://developer.mozilla.org/en-US/docs/Tools/Scratchpad
  - JS Bin : http://jsbin.com
- Documentation de qualité sur JavaScript et DOM (regarder la version anglaise de préférence) :
  - https://developer.mozilla.org/en-US/docs
- Gestion des événements en JavaScript :
  - http://www.alsacreations.com/article/lire/578-La-gestion-des-evenements-en-JavaScript.html
- Compatibilité des navigateurs
  - Avec ECMAScript 5 : http://kangax.github.com/es5-compat-table/
  - Avec DOM :
    - http://www.quirksmode.org/compatibility.html
    - http://en.wikipedia.org/wiki/Comparison_of_layout_engines_%28Document_Object_Model%29

JavaScript
Langage de programmation

Un premier exemple : x^n

Important pour un code de meilleure qualité :
erreurs générées si utilisation de mauvais patterns
☛ http://ejohn.org/blog/ecmascript-5-strict-mode-json-and-more/

```
"use strict";

var continuer;
do {
    var x = prompt("x ?");
    var n = prompt("n ?");

    var result = 1; // variable pour le resultat

    for(var i = 0; i < n; i++){
        result *= x; // x*x*x... n fois
    }

    alert(x+"^"+n+" = "+result); // affichage du resultat

    continuer = confirm("Voulez-vous continuer ?");
} while(continuer);
```

Entrées / sorties

- Interaction avec l'utilisateur via les fonctions prédéfinies du navigateur

```
alert("Message à l'utilisateur");
var donnee = prompt("Saisir une donnée :");
var bool = confirm("Confirmez ou annulez.");
```

- Interaction avec la console JavaScript du navigateur

```
console.log("Ceci est un message de log.");
console.log("La variable x vaut : " + x);
```

- Interaction avec le contenu d'une page web (vu un peu plus tard)

Variables et typage

```
var num = 4;
var text = "Mon premier texte"; // ou 'Mon premier texte'
var isTrue = true;
```

- Typage faible/implicite/dynamique :
  - pas d'indication de type lors de la déclaration d'une variable
  - type déterminé dynamiquement en fonction de la valeur

- Types de base : number, boolean, string

```
console.log(typeof num);    // number
console.log(typeof isTrue); // boolean
```

- Et aussi : function, object, undefined

```
var myVar;
console.log(typeof myVar);  // undefined
```

Structures de contrôle

- Conditionnelles : if, else, else if, switch, opérateur ternaire

```
if(confirm("Etes-vous majeur ?")) {
    alert("Vous pouvez voter.");
} else {
    alert("Vous ne pouvez pas encore voter.");
}
```

- Boucles : for, while, do-while

```
var i = 3;
while (i >= 0) {
    console.log(i);
    i--;
}
// affiche : 3 2 1 0
```

```
for (var i = 0; i <= 3; i++) {
    console.log(i);
}
// affiche : 0 1 2 3
```


LEARN
Exercice 1.1

Un autre exemple : tri d'un tableau

```
function triBulle(tab){
    var echange;
    do {
        echange = false;
        for(var i = 0; i < tab.length-1 ; i++) {
            if(tab[i] > tab[i+1]) {
                var tmp = tab[i];
                tab[i] = tab[i+1];
                tab[i+1] = tmp;
                aucunEchange = true;
            }
        }
    } while (echange);
}

var monTab = [9,6,8,7,10,1,3,4,2,5];
triBulle(monTab);
console.log(monTab); // [1,2,3,4,5,6,7,8,9,10]
```

Fonctions

- Définition d'une fonction

```
function surfaceRectangle(longueur, largeur) {
    return longueur*largeur;
}
```

- Appel d'une fonction

```
surfaceRectangle(2.5, 3); // 7.5
```

- Les fonctions sont identifiées par leur nom 

- Fonctions prédéfinies
  - Liées au navigateur : alert(), prompt()…
  - Globales : parseInt(), parseFloat(), isNaN()…

Portée des variables

- Pas de notion de bloc…
- Les variables qui sont déclarées à l'intérieur d'une fonction sont *locales* à cette fonction

Quel que soit l'endroit où est déclarée la variable dans la fonction, sa portée est l'ensemble de la fonction !

```
var solde = 10;

function debit(montant){
    if (montant > solde) {
        var agio = 5;
    }
    solde = solde - montant - agio;
    return solde;
}
```

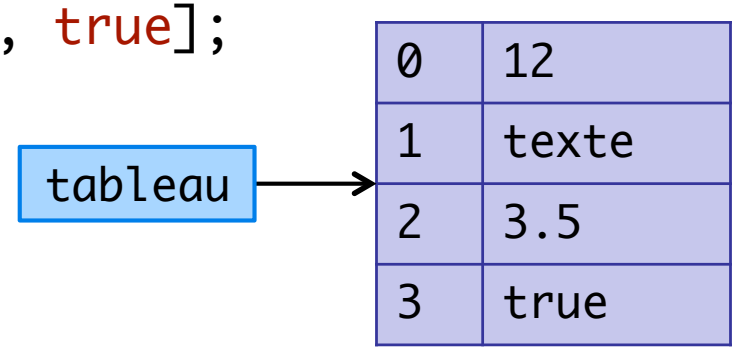
globale

locale, portée : fonction debit

ici, agio peut ne pas avoir été définie…
dans ce cas ➡ solde = NaN

Tableaux

```
var tableau = [12, "texte", 3.5, true];
console.log(tableau);
console.log(tableau[0]);
console.log(tableau.length);
```



- Les tableaux sont en fait des objets

```
console.log(typeof tableau); // object
```

- Autre façon de créer un tableau

```
var tableau = new Array(12, "texte", 3.5, true);
```

- Exemples de méthodes prédéfinies des tableaux

```
var tab = [1,2,3];
tab.pop();     // [1,2]
tab.push(4);   // [1,2,4]
tab.reverse(); // [4,2,1]
```


LEARN
Exercice 1.2

Quoi d'autre sur les fonctions ?

- Passage des paramètres à une fonction
  - number, boolean, string : copie de la valeur
  - object, function : copie de la référence

```
var monTab = [9,6,8,7,10,1,3,4,2,5];
triBulle(monTab); // modifie monTab !
```

- Les fonctions sont en fait des « objets »… !

```
function add(x1, x2) {
    return x1+x2;
}
console.log(typeof add); // function
console.log(add);
// function add(x1, x2) {return x1+x2;}

console.log(add(1,2)); // 3
```

affiche la fonction

affiche le résultat de l'appel de la fonction

Fonctions lambda (anonymes)

- Exécution immédiate

```
(function (longueur, largeur) {
    return longueur*largeur;
}) (2.5, 3); // 7.5
```

Fonction définie et appelée en même temps

- Stockage dans une variable

Equivalent avec la façon "classique" de définir une fonction

```
var surfaceRectangle = function (longueur, largeur) {
    return longueur*largeur;
}
console.log(typeof surfaceRectangle); // function
console.log(surfaceRectangle(2.5, 3)); // 7.5
```

Fonction en paramètre d'une fonction

- Une fonction est un « objet », par conséquent elle peut être passée en paramètre à une autre fonction !

```
function applique(f, v) {
    return f(v);
}

applique(Math.sin, 3.1416); // -0.0000734
```

Appelle la fonction f sur la variable v

$x \longmapsto \sin(x)$

- Utilisation avec une fonction anonyme :

```
applique(function(x){return x*x;}, 3.1416); // 9.86965
```

Fonction anonyme
$x \longmapsto x^2$


LEARN
Exercice 1.3

Problème de géométrie

- Dans une application de dessin :
  - On représente des formes géométriques par les coordonnées de leurs sommets dans un repère
  - On souhaite pouvoir translater les formes
- Une implémentation possible :

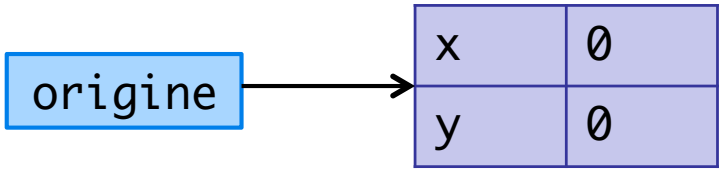
```
var triangle = [[0,0],[0,3],[3,0]];

function translaterForme(forme, dx, dy){
    for(var i = 0; i < forme.length ; i++){
        forme[i][0] += dx;
        forme[i][1] += dy;
    }
}

translaterForme(triangle,1,1);
console.log(triangle); // [[1, 1], [1, 4], [4, 1]]
```

Problème(s) :
conventions implicites ➡ rien ne garantit l'ordre des coordonnées…

Premiers objets

- Objet défini par des littéraux

```
var origine = {
    x: 0,
    y: 0
};
console.log(typeof origine); // "object"
console.log(origine);        // ({x:0,y:0})
console.log(origine.y);      // 0
console.log(origine["y"]);   // 0
```



Deux façons d'accéder aux attributs : avec . et avec []
➥ objet ≈ tableau associatif

- Boucle *for* simplifiée sur les attributs d'un objet

```
for (var attr in origine) {
    console.log(origine[attr]);
}
```

Notre problème de géométrie

```
var triangle = [
    {x:0,y:0},
    {x:0,y:3},
    {x:3,y:0}
];

function translaterForme(forme, dx, dy){
    for(var i = 0; i < forme.length ; i++){
        forme[i].x += dx;
        forme[i].y += dy;
    }
}

translaterForme(triangle,1,1);
console.log(triangle);
// [{x:1, y:1}, {x:1, y:4}, {x:4, y:1}]
```


LEARN
Exercice 1.4

Notre problème de géométrie

```
var triangle = [
    {x:0,y:0},
    {x:0,y:3},
    {x:3,y:0}
];

function translaterForme(forme, dx, dy){
    for(var i = 0; i < forme.length ; i++){
        forme[i].x += dx;
        forme[i].y += dy;
    }
}

translaterForme(triangle,1,1);
console.log(triangle);
// [{x:1, y:1}, {x:1, y:4}, {x:4, y:1}]
```

Problème :

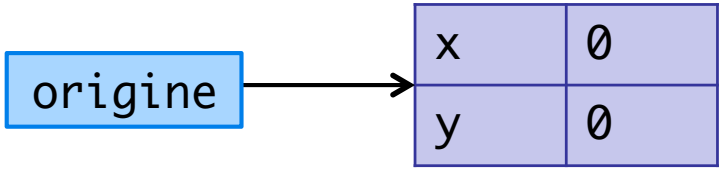
rien ne garantit que les coordonnées des points porteront toujours le nom x et y…

« Modèle » d'objet

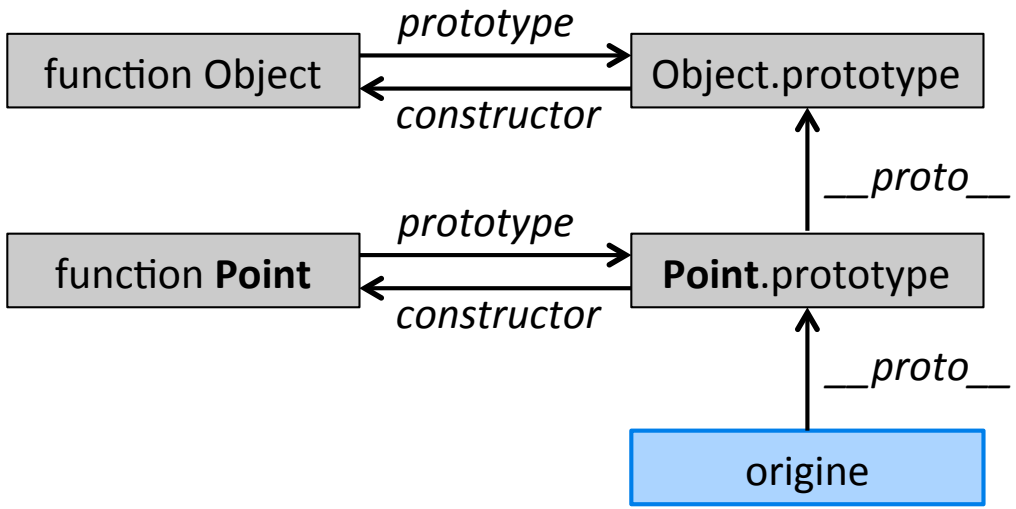
- Constructeur d'objet :

```
function Point(x,y){
    this.x = x;
    this.y = y;
}
var origine = new Point(0,0);
console.log(origine);
```

Tous les objets construits avec ce constructeur ont les mêmes attributs (avec des valeurs différentes)



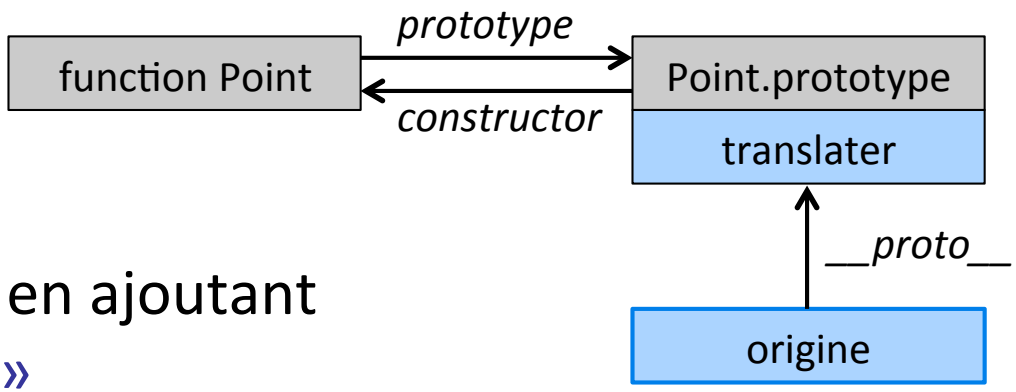
- Prototype : objet, qui sert de « modèle » d'objet



Objets avec des méthodes

- Définition d'un constructeur de point

```
function Point(x,y){
    this.x = x;
    this.y = y;
}
```



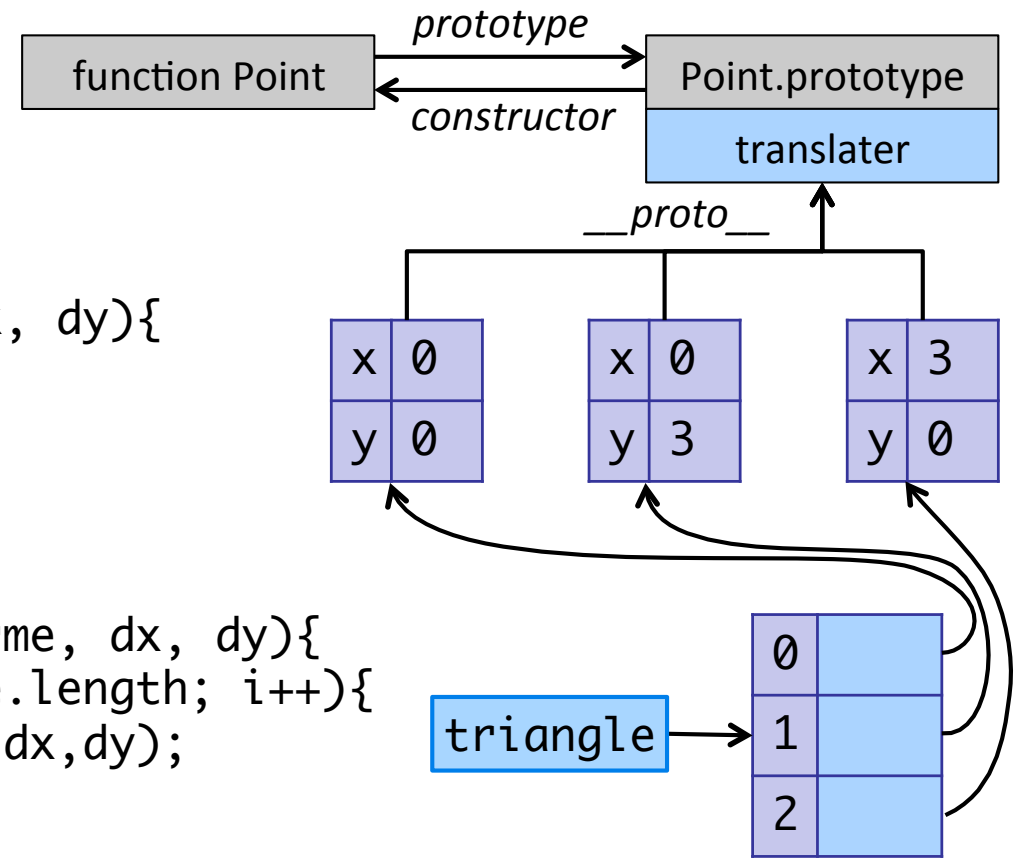
- Redéfinition du prototype en ajoutant une méthode « translater »

```
Point.prototype = {
    constructor: Point,
    translater: function (dx, dy){
        this.x += dx;
        this.y += dy;
    }
}
```

Attention de garder la référence sur le constructeur dans le prototype

Notre problème de géométrie

```
function Point(x,y){
    this.x = x;
    this.y = y;
}

Point.prototype = {
    constructor: Point,
    translater: function (dx, dy){
        this.x += dx;
        this.y += dy;
    }
}

function translaterForme(forme, dx, dy){
    for(var i = 0; i < forme.length; i++){
        forme[i].translater(dx,dy);
    }
}

var triangle = [new Point(0,0), new Point(0,3), new Point(3,0)];
translaterForme(triangle,1,1);
console.log(triangle); // [{x:1, y:1}, {x:1, y:4}, {x:4, y:1}
```



Quelques « modèles » d'objets existants

- String
- Number
- Boolean
- Date
- Error
- Array
- Iterator
- RegExp
- Math
  - Math.PI
  - Math.E
  - Math.sin(x)
  - Math.sqrt(x)
  - Math.pow(x,n)
  - …

☛ **Voir :**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference


LEARN
Exercice 1.5

JavaScript
Pour dynamiser des pages web

Inclusion de JavaScript dans une page web

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Inclusion de code JavaScript</title>
    <script type="text/javascript">
// <![CDATA[

"use strict";
console.log("dans la page web");

// ]]>
    </script>
  </head>
  <body>
    <h1>Bienvenue sur ma page...</h1>
  </body>
</hml>
```

Inclusion du code directement dans la page

`<![CDATA[` ➡ code JavaScript non considéré comme du HTML

`//` ➡ balise CDATA mise en commentaire pour JavaScript

Inclusion de JavaScript dans une page web

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Inclusion de code JavaScript</title>
    <script type="text/javascript" src="script.js"></script>
  </head>
  <body>
    <h1>Bienvenue sur ma page...</h1>
  </body>
</hml>
```

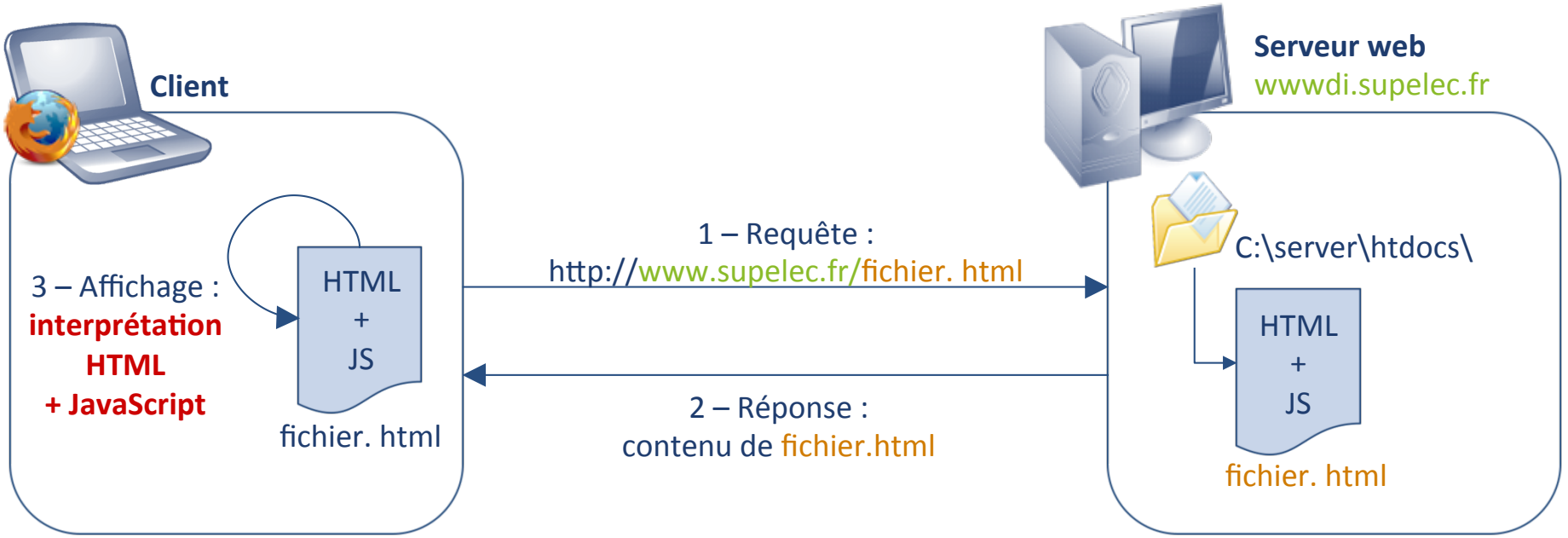
Inclusion d'un fichier contenant le code

fichier script.js

```
"use strict";
console.log("dans un fichier");
```

Exécution du code ?

Côté client,
dans le navigateur



Exécution du code ?

Pendant que le navigateur charge la page

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Inclusion de code JavaScript</title>
    <script type="text/javascript">
// <![CDATA[

"use strict";
console.log("dans la page web");

// ]]>
    </script>
  </head>
  <body>
    <h1>Bienvenue sur ma page...</h1>
  </body>
</hml>
```


LEARN
Exercice 2.1

Modèle événementiel

- Evènement : changement d'état de l'environnement qui peut être intercepté par JavaScript

- Quand sont produits des événements dans une page web ?
  - Dès qu'il se passe quelque-chose dans la fenêtre du navigateur
  - Lorsque l'utilisateur interagit avec des éléments HTML via la souris ou le clavier

Souris	Clavier	Formulaire	Fenêtre / objet HTML
`click`	`keydown`	`focus`	`load`
`dblclick`	`keyup`	`blur`	`unload`
`mousedown`	`keypress`	`select`	`abort`
`mouseup`		`change`	`resize`
`mouseover`		`submit`	`scroll`
…		`reset`	…

Déclencher l'exécution de JavaScript

- Sur un clic de souris :

```
<span onclick="console.log('Hello !');">
    Cliquez-moi !
</span>
```

Gestionnaire d'événements de type « click » : onclick

- Sur modification d'un champ de formulaire :

```
<form>
    <input type="text"
           value="default text"
           onchange="alert('Texte changé !');" />
</form>
```

- A la fin du chargement complet de la page web :

```
…
<body onload="console.log('Page web chargée.');" >
    …
</body>
```

Intercepter des événements

- Certains éléments HTML réagissent déjà à des événements liés à la souris ou au clavier : liens, boutons, champs…
  ☛ Il est possible d'intercepter ces événements pour exécuter un traitement, puis de :
  - Laisser le traitement par défaut de l'événement se poursuivre

```
<a href="http://www.supelec.fr"
   onclick="alert('Clic !');">
       Cliquez-moi !
</a>
```

Le message sera affiché puis la page www.supelec.fr sera chargée

  - Ou interrompre le traitement par défaut de l'événement

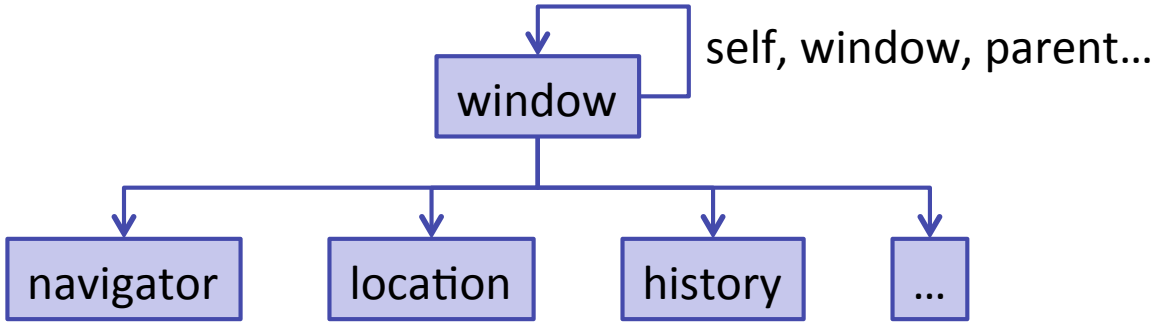
```
<a href="http://www.supelec.fr"
   onclick="console.log('Clic !'); return false;">
       Cliquez-moi !
</a>
```

La page www.supelec.fr ne sera pas chargée


LEARN
Exercice 2.2

Le contexte « window »

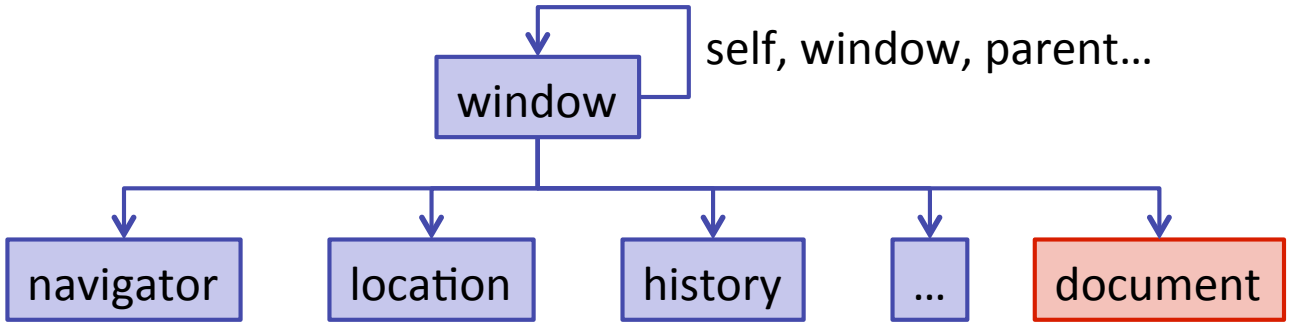
- JavaScript s'exécute dans une fenêtre de navigateur web, qui est représentée par un objet appelé « window », disponible dans tout programme JavaScript
  - Sert de contexte d'exécution global et de conteneur
  - Fournit des fonctions d'interactions (alert, prompt…)




```
console.log(window.location);

window.alert("Interaction avec l'utilisateur");
// est plus simplement écrit : alert("…");
```

Accéder au contenu de la page (document)

- window permet d'accéder au contenu de la page web affichée dans la fenêtre du navigateur via un objet « document »



- L'objet document n'est disponible qu'une fois que la page est complètement chargée ➡ après l'événement load

```
window.onload = function (){
    // ici, code exécuté une fois la page chargée
    console.log(window.document);
};
```

Attendre que la page soit chargée
(événement load) avant d'accéder à son contenu

Fonction « attachée » à un événement

```
<!DOCTYPE html>
<html>
  <head> …
    <script type="text/javascript">
// <![CDATA[

"use strict";

function myFunction(){
    console.log('hello');
};

// ]]>
    </script>
  </head>
  <body onload="myFunction();">
    <h1>Bienvenue… </h1>
  </body>
</hml>
```

```
<!DOCTYPE html>
<html>
  <head> …
    <script type="text/javascript">
// <![CDATA[

"use strict";

window.onload = function(){
    console.log('hello');
};

// ]]>
    </script>
  </head>
  <body>
    <h1>Bienvenue… </h1>
  </body>
</hml>
```

Deux façons différentes « d'attacher » une fonction à un événement, ici load


LEARN
Exercice 2.3

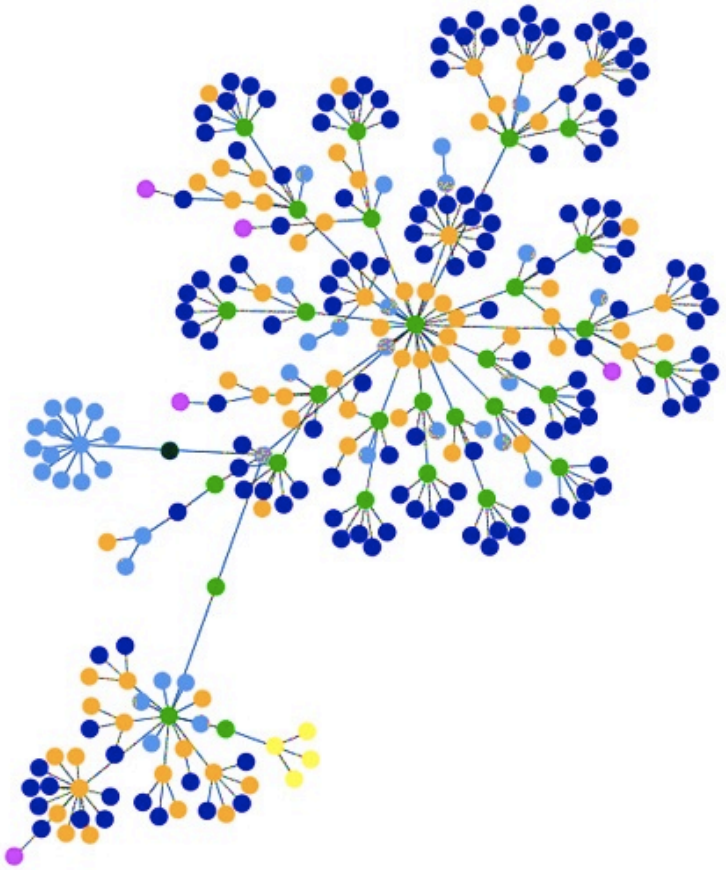
Modifier le contenu de la page (document)

- Il est possible de modifier le contenu de la page grâce aux méthodes et attributs disponibles sur l'objet « document »

- Ces méthodes et attributs implémentent en fait une interface standard de manipulation de documents XML et HTML appelée DOM (Document Object Model)
  - D'autres langages de programmation que JavaScript implémentent ce standard !

☛ Voyons ce qu'est DOM et comment l'utiliser !

DOM



Crédit : Haiko, http://www.aharef.info/2006/05/websites_as_graphs.htm

DOM : Document Object Model

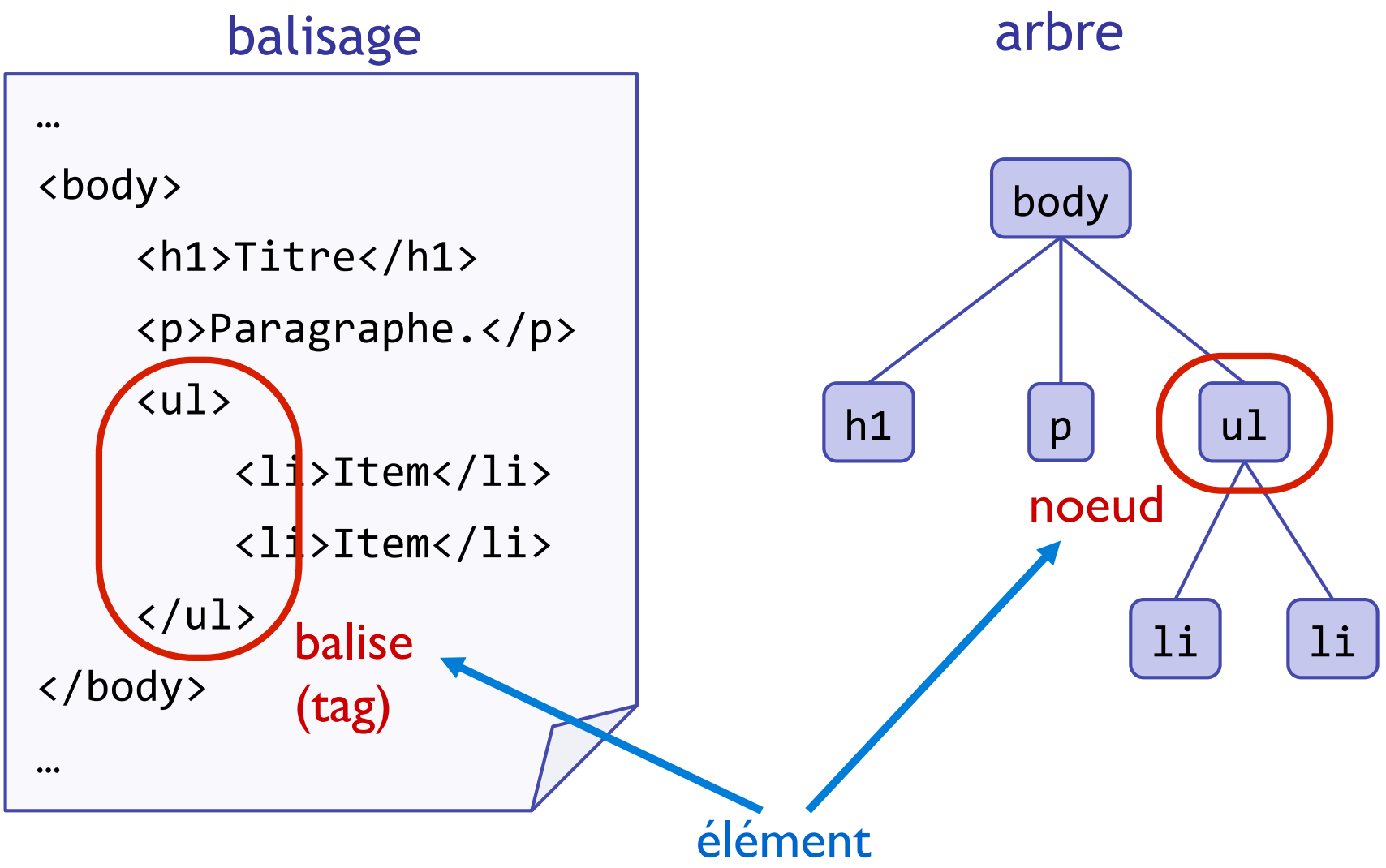
Documents XML/HTML vus sous forme d'arbre

Approche objet du modèle d'arbre : classes pour les nœuds, relations entre classes pour les arcs…

- DOM est conçu pour XML de façon générique
  + comporte des objets dédiés à HTML

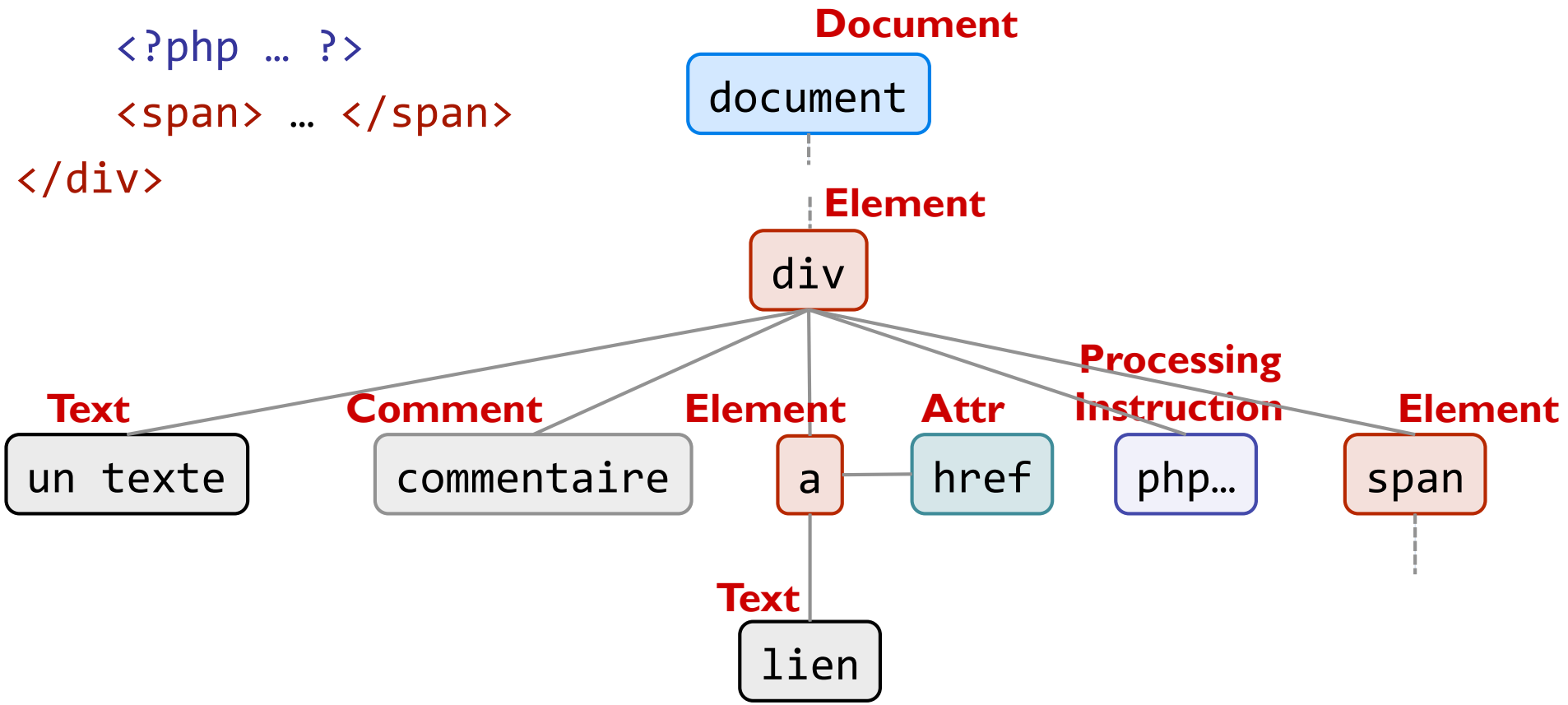
- DOM est un modèle abstrait, indépendant du langage :
  - Spécification officielle en IDL
  - Interfaces officielles pour Java et ECMAScript
  - Implémentations pour Java, JavaScript (navigateurs), et presque tous les autres langages

Dualité fichier HTML balisé / arbre

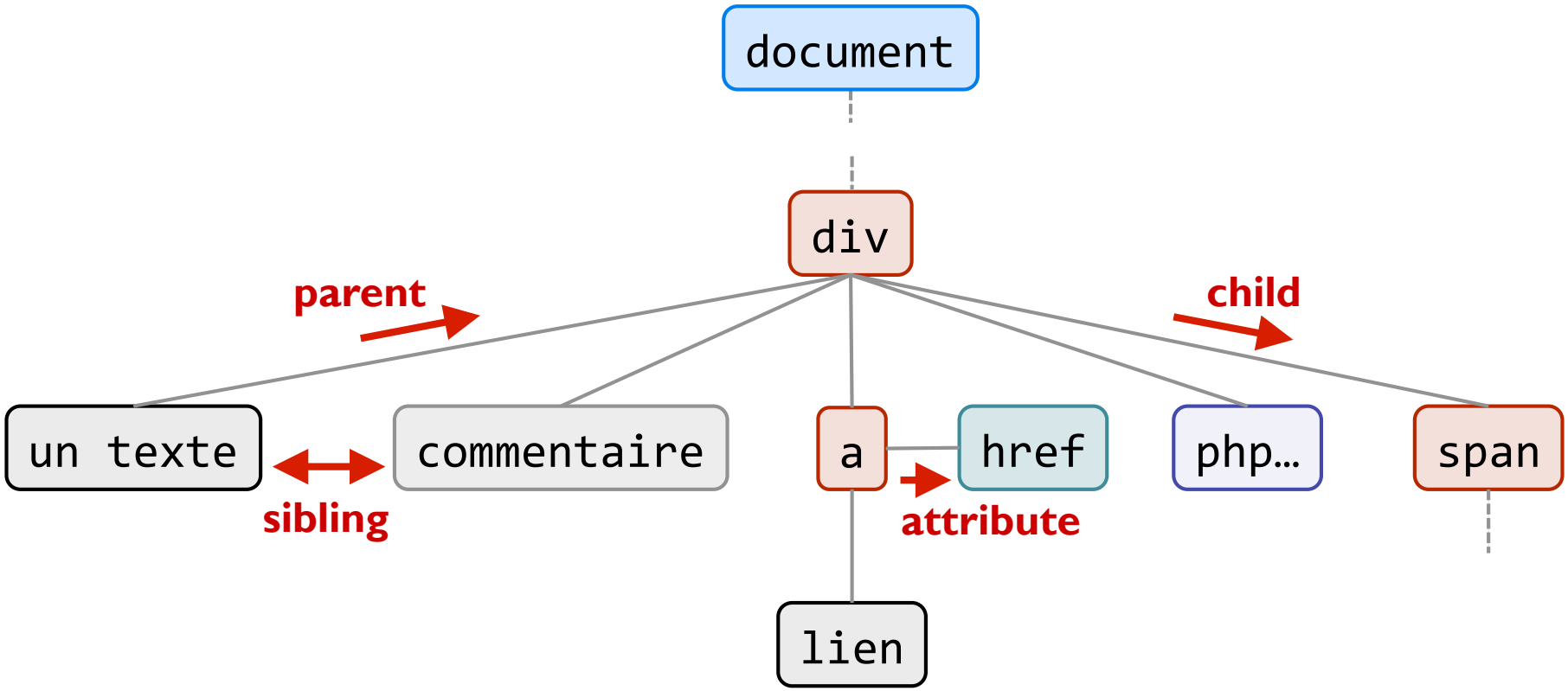


Principaux types de noeuds

```
<div>un texte
    <!-- commentaire -->
    <a href="url">lien</a>
    <?php … ?>
    <span> … </span>
</div>
```



Navigation dans l'arbre



Naviguer dans un document HTML avec JavaScript et DOM

A partir de l'objet document

- L'objet document propose des « raccourcis » pour accéder à certains éléments d'une page web
- A partir du nœud document « d » :

```
d.head : élément <head>
d.title : élément <title>

d.body : élément <body>

d.images : collection d'éléments <img>
d.links : collection d'éléments <a>
d.forms : collection d'éléments <form>
```


LEARN
Exercice 3.1

Navigation dans l'arbre DOM

- Sur n'importe quel nœud « n » :

  `n.nodeName` : *chaîne de caractères*
  `n.nodeType` : *entier (1: élément, 2: attribut, 3: texte, …)*
  `n.nodeValue` : *chaîne de caractères (pour nœuds attributs et texte)*

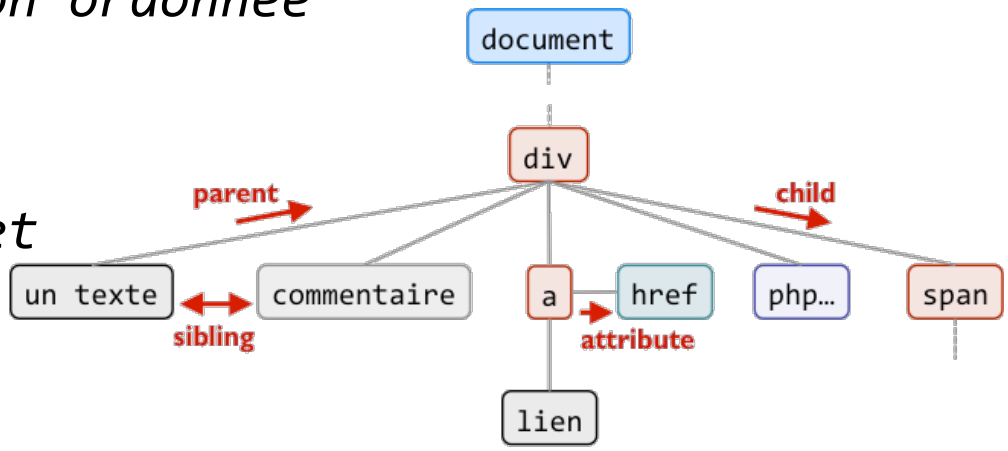
  `n.hasAttributes()` → *booléen*
  `n.attributes` : *collection ordonnée*

  `n.parentNode` : *objet*

  `n.hasChildNodes()` → *booléen*
  `n.childNodes` : *collection ordonnée*
  `n.firstChild` : *objet*
  `n.lastChild` : *objet*

  `n.previousSibling` : *objet*
  `n.nextSibling` : *objet*

  `n.ownerDocument` : *objet*



Exemple : navigation à partir de *document*

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Premier exemple avec DOM</title>
    <script type="text/javascript" src="monscript.js"></script>
  </head>
  <body>
    <h1>Bienvenue sur ma page...</h1>
  </body>
</hml>
```

```js
"use strict";
window.onload = function (){

    console.log(window.document.nodeName); // "#document"
    console.log(window.document.hasChildNodes()); // true
    console.log(window.document.firstChild.nodeName); // "html"

};
```

Rappel : attendre que la page soit chargée avant d'accéder à son contenu

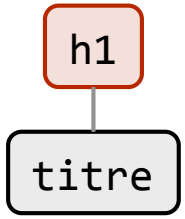
Recherche dans l'arbre DOM

- A partir du nœud document « d » :

  `d.getElementById(idOfElement)` → *élément portant l'identifiant donné*

  `d.getElementsByTagName(tagName)` → *collection d'éléments portant le tag donné*

- Exemple : recherche + navigation




```
"use strict";
window.onload = function (){
    var elements = window.document.getElementsByTagName("h1");
    for(var i = 0; i < elements.length; i++){
        console.log(elements[i].nodeName); // "H1"
        console.log(elements[i].firstChild.nodeValue);
    }
};
```

Premier enfant du nœud h1 : c'est un nœud texte dont la valeur est l'intitulé du titre

A propos des nœuds éléments

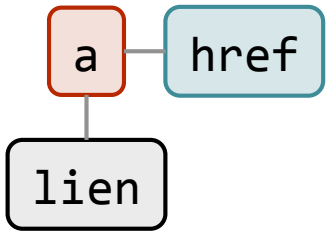
- A partir d'un nœud de type élément « e » :

  `e.id` : *chaîne de caractères*

  `e.hasAttribute(name)` → *booléen*
  `e.getAttribute(name)` → *chaîne de caractères*
  `e.setAttribute(name, value)`
  `e.removeAttribute(name)`

  `e.getElementsByTagName(tagName)` → *collection d'éléments* ***descendants de e*** *et portant le tag donné*

- Exemple : recherche + accès aux attributs




```
var elements = window.document.getElementsByTagName("a");
for(var i = 0; i < elements.length; i++){
    console.log(elements[i].getAttribute("href")); // url
}
```


LEARN
Exercice 3.2

Modifier un document HTML avec JavaScript

Création de nœuds

- A partir du nœud document « d » :

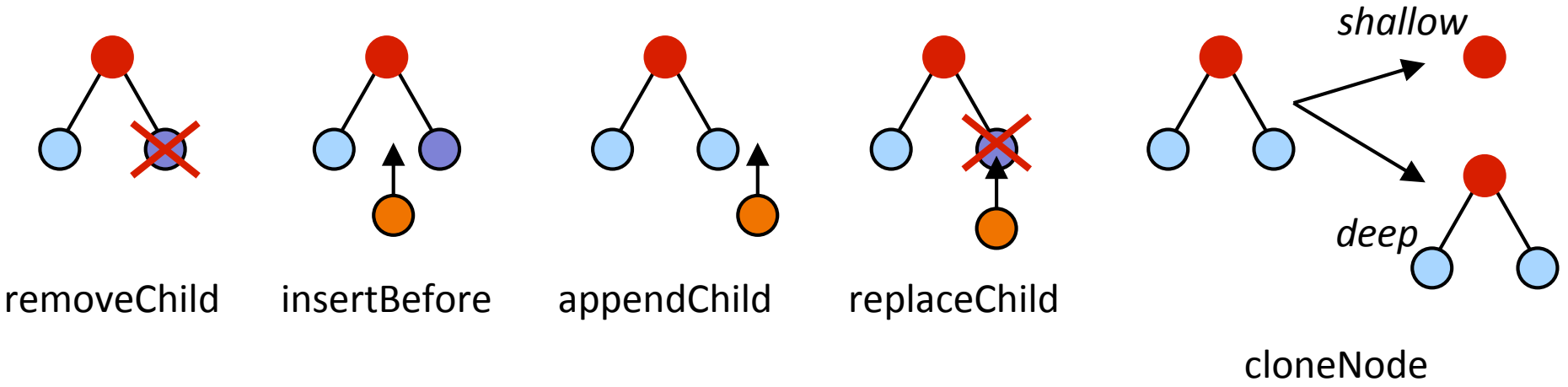
```
d.createElement(tagName) → élément créé
d.createElementNS(namespaceURI, tagName) → élément créé

d.createTextNode(text) → nœud texte créé
d.createComment(text) → nœud commentaire créé
d.createCDATASection(data) → nœud CDATA créé
d.createProcessingInstruction(target, data) → nœud PI cré

d.createAttribute(name) → nœud attribut créé
…
```

Attention :
il faut ensuite insérer les nœuds créés dans l'arbre !
☛ *voir slide suivant*

Modification de l'arbre DOM

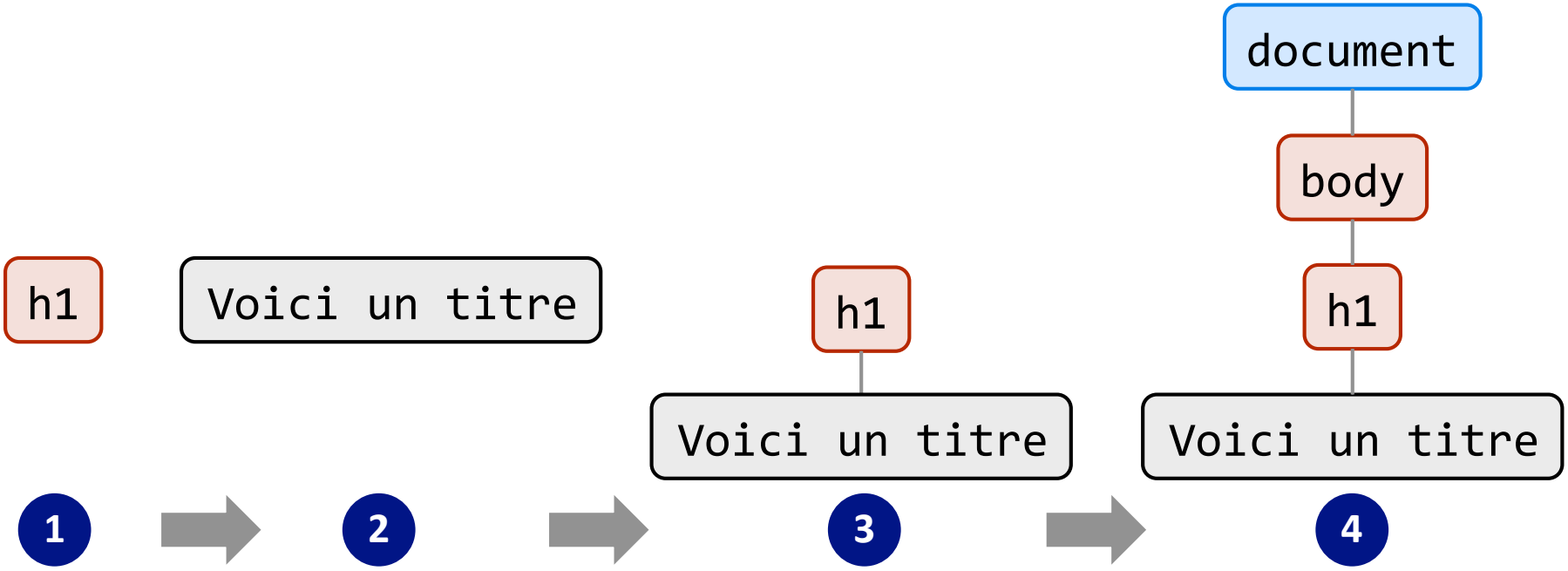
- Sur n'importe quel nœud « n » :

```
n.removeChild(nodeToRemove) → objet supprimé
n.insertBefore(nodeToInsert, childRef) → objet inséré
n.appendChild(nodeToAppend) → objet inséré
n.replaceChild(newNode, oldNode) → objet remplacé
n.cloneNode(deepOrNot) → objet résultat de la copie
```



Exemple : création d'un nœud <h1>

```
var d = window.document;

1  var noeudH1 = d.createElement("h1");
2  var noeudTexte = d.createTextNode("Voici un titre");

3  noeudH1.appendChild(noeudTexte);

4  d.body.appendChild(noeudH1);
```




LEARN
Exercice 3.3

Exercice d'application JavaScript + DOM

Conclusion

En résumé

- JavaScript = langage de programmation conçu pour écrire des scripts interprétés par un navigateur web
  - impératif avec typage faible/implicite/dynamique
  - orienté objet « à prototype »
  - fonctionnel

- DOM = représentation arborescente des documents XML/HTML + APIs de navigation et de modification de l'arbre
  - Modèle abstrait indépendant du langage
  - Différentes implémentation de processeurs DOM

Conclusion

- JavaScript + DOM = pages web dynamiques !
- Et nous verrons ensuite comment Jquery facilitera énormément l'écriture de vos programmes JavaScript !

- Petits bémols :
  - Il est possible de faire *n'importe quoi* en JavaScript…
  - Trop de JavaScript tue le JavaScript !
  - Problèmes de compatibilité entre navigateurs (JS & DOM)

- JavaScript = scripts interprétés par un navigateur web
  … mais pas que ?
  - JSON, JavaScript côté serveur (Node.js),
    JavaScript dans Java (Rhino), émulateur de PC en JavaScript…