Séance 08 :

→ REACT.JS – Introduction au **JavaScript**

INF37407 – Technologie de l'inforoute

Prof. **Yacine YADDADEN**, Ph. D.

UQAR

Plan

1. **Introduction au JavaScript**
    1. Introduction au rôle du JavaScript dans le Web
    2. Intégration d’un script dans une page Web
    3. Bases du langage JavaScript (ES6)
    4. Manipulation du DOM et gestion d’événements
2. **Introduction au TypeScript**
3. Questions & Discussion

UQAR

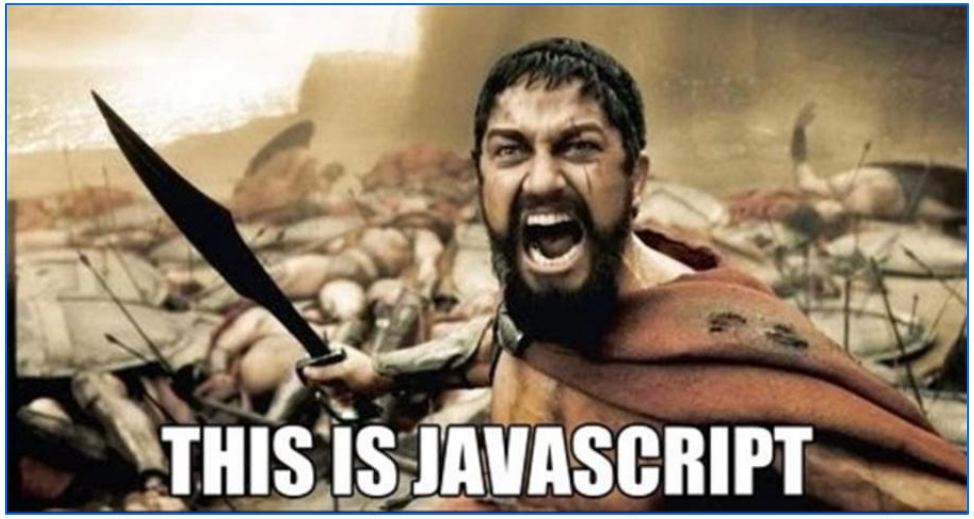
Partie I

Introduction au rôle du JavaScript dans le Web

UQAR

Introduction au rôle du JavaScript dans le Web

→ JavaScript, quésaco ?



- C’est le langage de *script* par excellence des navigateurs Web,
- Il offre la possibilité d’implémenter des *traitements* élaborés dans des pages Web,
- Il peut être mis en œuvre dans toutes application disposant d’un *interpréteur* pour ce langage.

UQAR

Introduction au rôle du JavaScript dans le Web

→ Historique

- Il a été créé par Brendan Eich pour le compte de Netscape Communication Corporation,
- En novembre 1996, il a servi de fondation à la première version du standard ECMA-262 décrivant le langage ECMAScript,
- JavaScript correspond donc à un dialecte de ce standard et a évolué indépendamment par la suite.
- **ECMA :** *« Fondé en 1961 et à vocation internationale depuis 1994, ECMA est un organisme de standardisation pour les systèmes d'information et de communication. Son objectif est de développer et promouvoir divers standards afin de faciliter l'utilisation des technologies de l'information et de la communication. »,*
- **ECMAScript :** *« Fondé sur la spécification ECMA-262, ECMAScript consiste en un langage interprété, dont l'objectif est de permettre la manipulation d'objets fournis par l'application hôte. ».*

UQAR

Introduction au rôle du JavaScript dans le Web

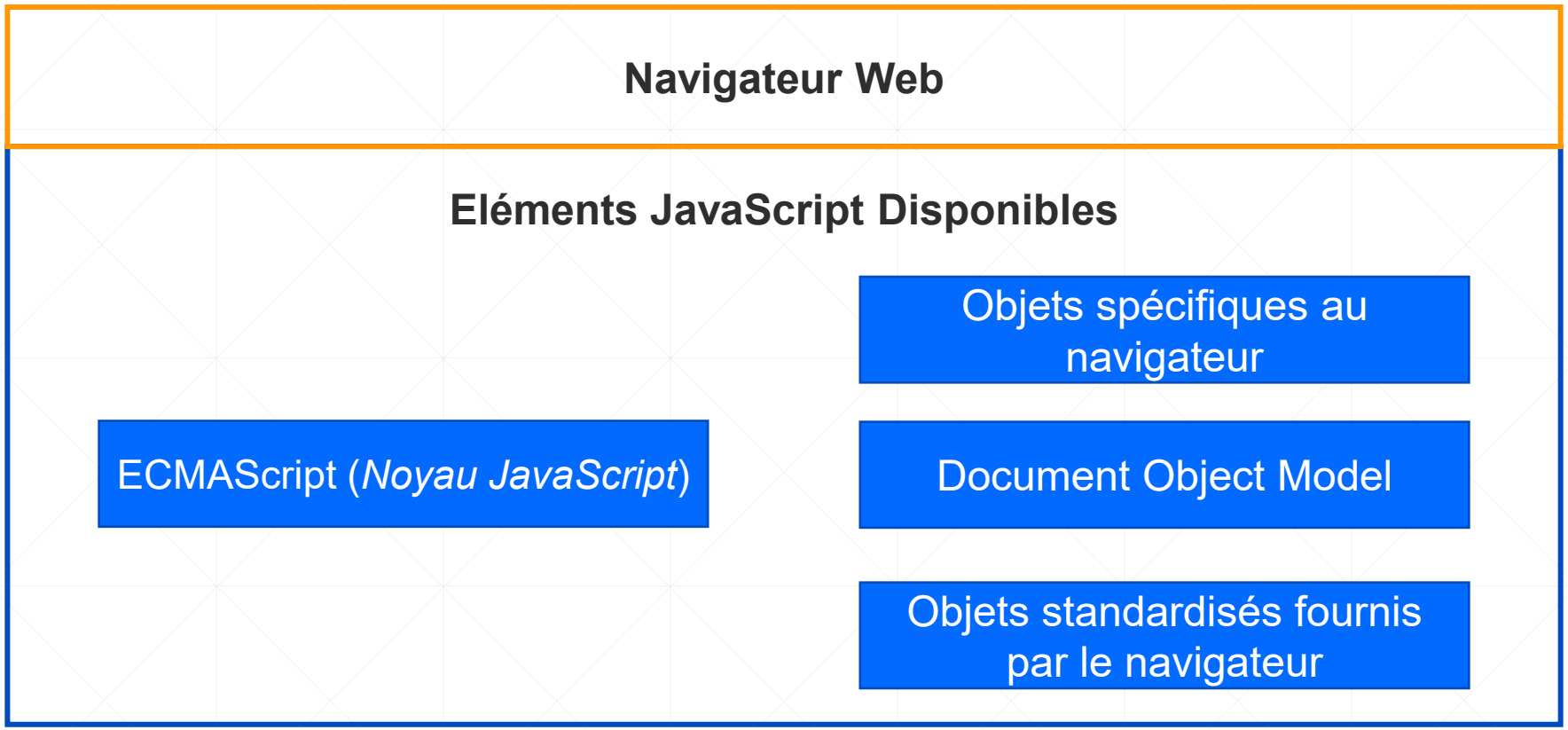
→ Le DOM

- Ces dialectes enrichissent l'ensemble des objets de base du langage ECMAScript avec d'autres objets provenant de l'environnement d'exécution du script. Cet environnement peut correspondre à un navigateur Web ou à un interpréteur JavaScript embarqué dans une application.
- Ces objets peuvent être normalisés, comme c'est quasiment le cas de ceux relatifs à la technologie **DOM** (*Document Object Model*), ou être spécifiques du navigateur.
- **DOM** : *« DOM correspond à une représentation en mémoire d'un arbre XML normalisée sous forme d'objets. Dans le cas de pages HTML, cette technologie offre la possibilité de manipuler leur structure en mémoire par programmation. ».*

UQAR

Introduction au rôle du JavaScript dans le Web

→ Eléments JavaScript



UQAR

Introduction au rôle du JavaScript dans le Web

→ Rôle et Objectif

☑ *JavaScript* propose différents mécanismes permettant de faire interagir les éléments écrits avec les langages *HTML* et *CSS* d'une page Web avec les scripts JavaScript,

☑ **HTML :** *« ou HyperText Markup Language est un langage de balisage utilisé afin de mettre en oeuvre des pages Web contenant notamment des liens et des éléments graphiques. Ces pages sont conçues pour être affichées dans des navigateurs Web. »,*

☑ **CSS :** *« ou Cascading Style Sheets adresse les problématiques d'affichage des pages HTML. Son objectif est de permettre une séparation claire entre la structure d'une page, par le biais du langages HTML et sa présentation, décrite avec CSS. ».*

UQAR

Introduction au rôle du JavaScript dans le Web

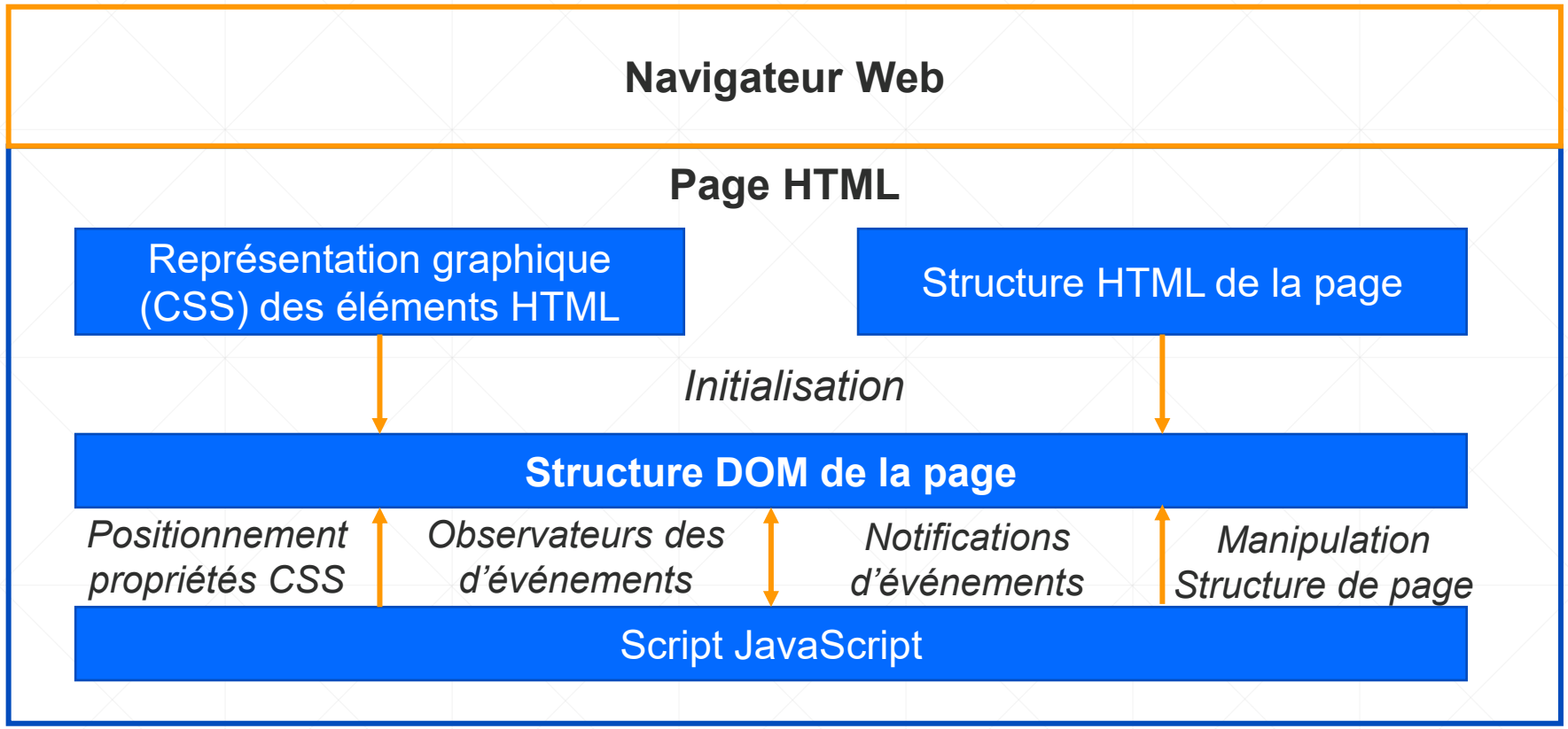
→ Constituants page Web & leurs interactions

Structure de représentation initiale de la page

Structure de représentation en mémoire

Interaction avec la structure de la page

Traitements de la page



UQAR

Introduction au rôle du JavaScript dans le Web

→ Avantages

☑ En se fondant sur les technologies HTML et CSS, JavaScript offre la possibilité de réaliser des interfaces graphiques de pages Web plus élaborées et plus interactives,

☑ Le support amélioré de la technologie CSS par les navigateurs a également grandement favorisé cet enrichissement,

☑ **Interactivité :** *« offre la possibilité de déclencher des traitements suite à des événements utilisateur par le biais de la technologie DOM. Les entités JavaScript enregistrées pour l'événement sont alors déclenchées afin de réaliser des traitements. »,*

☑ **Prise en compte des navigateurs :** *« La plupart des navigateurs du marché possèdent différentes spécificités quant à l'utilisation du DOM, de CSS et de JavaScript. La difficulté consiste donc à assurer le même rendu dans différents navigateurs. ».*

UQAR

Introduction au rôle du JavaScript dans le Web

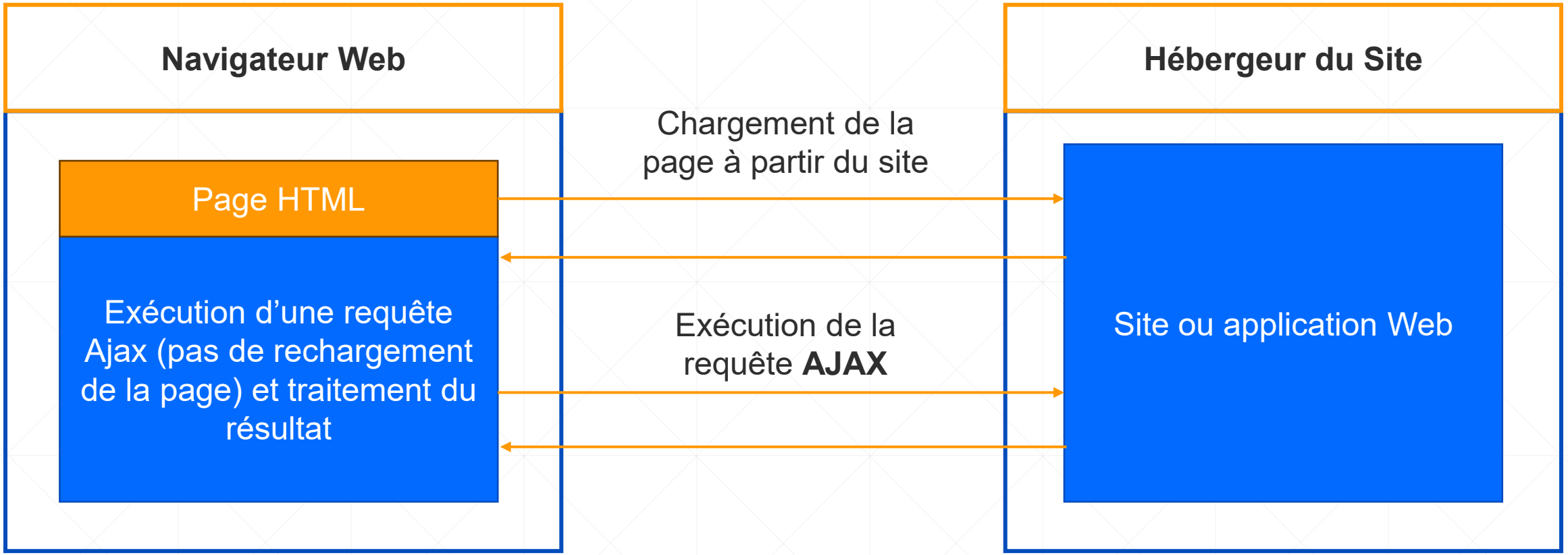
→ Échange des données

- JavaScript permet de réaliser des requêtes **HTTP** en recourant aux mécanismes fournis par le langage JavaScript. La plupart des navigateurs récents supportent désormais cette fonctionnalité,
- **HTTP :** *« ou HyperText Transfer Protocol correspond au protocole de transport applicatif de base d'Internet. Il s'agit d'un protocole ASCII transportant des informations sous forme de texte et qui peut fonctionner sur n'importe quel protocole réseau fiable, le plus souvent TCP. »,*

UQAR

Introduction au rôle du JavaScript dans le Web

→ Échange des données



UQAR

Partie II

Intégration d’un script dans une page Web

UQAR

Intégration d'un script dans une page Web

→ Structuration des applications

- **Isolation des langages :**
  - Dans la mesure où les pages Web peuvent recourir à différents langages, il est important de bien séparer leur utilisation et de soigner leurs interactions.
  - Les styles CSS et le code JavaScript doivent être le plus possible externalisés de la des balises HTML.
- **Structuration des traitements JavaScript**
  - Divers mécanismes peuvent être mis en œuvre et qui s'appuient essentiellement sur les fonctions JavaScript. Ils permettent notamment d'exploiter les concepts de la POO.
  - L'objectif est de modulariser les traitements dans des entités JavaScript afin d'éviter les duplications de code et de favoriser la réutilisation des traitements.

UQAR

Intégration d'un script dans une page Web

→ Exécution de Scripts JavaScript

- JavaScript n'étant pas un langage compilé, il doit être exécuté par le biais d'un interpréteur, dont plusieurs sont disponibles gratuitement tandis que d'autres sont intégrés aux navigateurs Web.
- Des interpréteurs JavaScript sont disponibles dans différents produits du marché. Ils peuvent correspondre à des implémentations propriétaires ou à des incorporations de moteurs d'interprétation externes.
- Il y a deux façons d'exécuter du code JavaScript sur une page Web :
  1. Ecrire du code sur la page elle-même,
  2. Ou bien faire appel à un fichier externe.

UQAR

Intégration d'un script dans une page Web

→ Exécution de Scripts JavaScript

```
<!DOCTYPE html>
   <html>
      <head>
         <script type="text/javascript" src="script.js"></script>
      </head>
      <body>

      </body>
</html>
```

UQAR

Intégration d'un script dans une page Web

→ Exécution de Scripts JavaScript

```
<!DOCTYPE html>
   <html>
      <head>

      </head>
      <body>
         <script type="text/javascript">
            /* code JavaScript */
         </script>
      </body>
</html>
```

UQAR

Intégration d'un script dans une page Web
→ Lire & Afficher

- Afin d'afficher des données à travers une fenêtre *pop-up*, on utilise le code :

```
alert("Texte à afficher !");
alert(maVariable);
```

- Afin de récupérer des données à travers une fenêtre *pop-up*, on utilise le code :

```
var maVariable = prompt("Entrer une valeur : ");
```

UQAR

Partie III

Bases du langage JavaScript (ES6)

UQAR

Bases du langage JavaScript (ES6)

→ Variables & Typage

- JavaScript est un langage non typé. Cela signifie que le type d'une variable est défini uniquement au moment de l'exécution,
- La mise en œuvre d'une variable se réalise par l'intermédiaire du mot-clé **var**. L'interpréteur JavaScript a la responsabilité de créer la valeur du bon type en fonction de l'initialisation ou de l'affectation,
- Le langage n'impose pas l'initialisation des variables au moment de leur création et offre la possibilité d'en définir plusieurs en une seule instruction.

```
// Variable initialisée avec une chaîne de caractères
var variable1 = "mon texte d'initialisation";
// Variable non initialisée
var variable2;
// Définition de plusieurs variables en une seule instruction
var variable3 = 2, variable4 = "mon texte d'initialisation";
```

UQAR

Bases du langage JavaScript (ES6)

→ Types primitifs de JavaScript

Type	Description
`Boolean`	*Type dont les valeurs possibles sont* ***`true`*** *et* ***`false`****.*
`Null`	*Une variable a été initialisée mais ne pointe sur aucun objet.*
`Number`	*Type qui représente un nombre.*
`String`	*Type qui représente une chaîne de caractères.*
`Undefined`	*Une variable définie possède cette valeur avant qu'elle soit initialisée.*

UQAR

Bases du langage JavaScript (ES6)

→ Détermination du type

- La méthode **`typeof`** permet de déterminer le type d'une variable. Elle renvoie la valeur **`object`** dans le cas de variables par référence, et ce quel que soit le type de l'objet,
- Voir le code ci-dessous :

```
// variable1 est de type undefined
var variable1;
alert(" type de variable1 : " + (typeof variable1));
// variable4 est de type string
var variable4 = "une chaîne de caractères";
alert(" type de variable4 : "+(typeof variable4));
```

UQAR

Bases du langage JavaScript (ES6)

→ Conversion de type

- JavaScript fournit des méthodes afin de convertir un type primitif en un autre. Il supporte les conversions de types primitifs en chaînes de caractères et de chaînes de caractères en nombres entiers ou réels.
- la méthode **`toString()`** qui permet de retourner la représentation des types primitifs sous forme de chaînes de caractères.
- La création de nombres à partir de chaînes de caractères se réalise par l'intermédiaire des méthodes **`parseInt()`** et **`parseFloat()`**.
- **NaN :** *« ou Not a Number correspond à un nombre dont la valeur est incorrecte. Elle se rencontre le plus souvent lors des conversions de types. ».*

UQAR

Bases du langage JavaScript (ES6)

→ Conversion de type

```
// variable2 contient la chaîne de caractère « 10 »
var nombreEntier = 10;
var variable2 = nomEntier.toString();
```

```
// entier1 contient le nombre 15
var entier1 = parseInt("15");

// reel contient le nombre réel 15,5
var reel = parseFloat("15.5");
```

UQAR

Bases du langage JavaScript (ES6)

→ Les opérateurs

Type	Description
Affectation	*Permet d'affecter une valeur ou une référence à une variable.*
Calcul	*Permet de réaliser les calculs de base sur les nombres.*
Comparaison	*Permet de réaliser des comparaisons entre différentes variables.*
Concaténation	*Permet de concaténer deux chaînes de caractères à l'aide de +.*
Conditionnel	*Permet d'initialiser la valeur d'une variable se basant sur une condition.*
Egalité	*Permet de déterminer si différentes variables sont égales.*
Logique	*Permet de combiner différents opérateurs de comparaison.*
Manipulation de bits	*Permet de manipuler des variables contenant des bits.*
Unaire	*Mot-clé du langage se fondant sur un seul élément.*

UQAR

Bases du langage JavaScript (ES6)

→ Structure de contrôle

- Le langage JavaScript définit plusieurs structures de contrôle afin de réaliser :
  - Des conditions,
  - Des boucles,
  - De mettre en œuvre des renvois.
- Ces structures classiques sont présentes dans la plupart des langages de programmation. Comme nous allons le voir, elles sont aisément utilisables.

Bases du langage JavaScript (ES6)

→ Conditions `if-else`

Il existe deux types de structures de conditions :

1. La première est fondée sur la combinaison d'instructions **`if`**, **`else if`** et **`else`**. Son rôle est de vérifier une condition et de réaliser différents traitements en cas de validité ou non. Il obéit à la syntaxe suivante :

```
if( condition ) {
    (...)
} else if( condition ) {
    (...)
} else {
    (...)
}
```

Bases du langage JavaScript (ES6)

→ Conditions `switch-case`

2. Le second type de structures permet de réaliser la même fonctionnalité que précédemment à l'aide de la combinaison d'instructions **`switch`**, case et default. Cela permet de mettre en œuvre des traitements en fonction de la valeur d'une variable. Il obéit à la syntaxe suivante :

```
switch( variable ) {
    case valeur1:
        (...)
        break;
    case valeur2:
        (...)
        break;
    default:
        (...)
}
```

Bases du langage JavaScript (ES6)

→ Boucles `for`

JavaScript définit quatre types de boucles :

1. La première s'appuie sur le mot-clé **`for`** afin de spécifier à la fois les traitements d'initialisation réalisés à l'entrée de la boucle, la condition de sortie et les traitements à réaliser après chaque itération. Tant que la condition de sortie est vraie, l'itération continue.

```
for( traitements d'initialisation ; condition de fin ;
    traitements à effectuer après chaque itération ) {
        (...)
    }
```

Bases du langage JavaScript (ES6)

→ Boucles `for ... in`

2. La boucle **`for ... in`** est une variante de la précédente qui se sert du mot-clé for conjointement avec le mot-clé in, ce dernier permettant de spécifier la variable à utiliser. Cette boucle permet de parcourir tous les éléments des tableaux indexés ou des objets, comme nous les verrons par la suite. Sa syntaxe est la suivante :

```
for( variable in structure ) {
    (...)
}
```

Bases du langage JavaScript (ES6)

→ Boucles `while`

3. La boucle **`while`** permet de spécifier la condition de fin. Aussi longtemps que cette condition est vraie, les traitements sont répétés. La syntaxe de cette boucle est la suivante :

```
while( condition de fin ) {
    (...)
}
```

Bases du langage JavaScript (ES6)

→ Boucles `do ... while`

4. La dernière boucle est une variante de la précédente, qui permet d'effectuer le bloc avant la première condition. La syntaxe de cette boucle est la suivante :

```
do {
    (...)
} while( condition de fin );
```

Bases du langage JavaScript (ES6)

→ Renvois

- JavaScript définit les mots-clés **`break`** et **`continue`** afin de modifier l'exécution des boucles :
- **`break` :** *« offre la possibilité d'arrêter l'itération d'une boucle et de sortir de son bloc d'exécution. »*,
- **`continue` :** *« offre la possibilité de forcer le passage à l'itération suivante. »*,
- Les traitements suivants de l'itération courante ne sont alors pas effectués.

Bases du langage JavaScript (ES6)

→ Méthodes de base

- JavaScript propose un ensemble de méthodes permettant de *détecter le type* et *la validité* de la variable passée en paramètre.

Méthode	Description
`Array.isArray()`	*Détermine si le paramètre est un tableau.*
`isNaN()`	*Détermine si la valeur du paramètre correspond à **NaN** (Not a Number).*

Bases du langage JavaScript (ES6)

→ Tableaux

Le langage JavaScript offre deux types de tableaux :

1. Tableau *classique :*

   « c'est une structure de données représentant une séquence finie d'éléments auxquels on peut accéder efficacement par leur position, ou indice, dans la séquence. »,

2. Tableau *associatif :*

   « il est aussi appelé dictionnaire est une structure de données associant à un ensemble de clefs un ensemble correspondant de valeurs. Chaque clef est associée à une valeur. ».

UQAR

Bases du langage JavaScript (ES6)

→ Tableaux *classiques*


```
// Initialisation d'un tableau vide
var tableau1 = [];
tableau1[0] = "1er élément";
tableau1[1] = "2eme élément";
// Initialisation d'un tableau avec des éléments
var tableau2 = ["1er élément", "2eme élément"];
```

UQAR

Bases du langage JavaScript (ES6)

→ Parcourir un tableau *classique*

Afin de parcourir un tableau *classique* et avoir accès à l'ensemble de ses éléments, on fait appel à une *boucle* **`for`**.

```
var tableau = [ "element1","element2" ];

...

for(var cpt = 0 ; cpt < tableau.length; cpt++) {
   alert("Elément pour l'index "+cpt+" : "+tableau[cpt]);
}
```

UQAR

Bases du langage JavaScript (ES6)

→ Méthodes associées au tableau *classique*

Méthode	Paramètre	Description
`concat()`	Tableau à ajouter	*Concatène un tableau à un autre.*
`join()`	Séparateur	*Construit une chaîne de caractère avec séparateur.*
`pop()`	-	*Supprime et retourne le dernier élément.*
`push()`	Élément à ajouter	*Rajoute un élément à la fin du tableau.*
`reverse()`	-	*Inverse l'ordre des élément d'un tableau.*
`shift()`	-	*Supprime et retourne le premier élément.*
`slice()`	Index de début et fin	*Retourne une partie du tableau.*
`sort()`	-	*Trie par ordre alphabétique.*
`toString()`	-	*Convertie en une chaîne de caractères.*
`unshift()`	Élément à ajouter	*Permet d'ajouter un élément au début du tableau.*

Bases du langage JavaScript (ES6)

→ Tableaux *associatifs*

Afin d'*initialiser* un tableau *associatif* et lui ajouter des données (*clef* et *valeur*) :

```
// Initialisation d'un tableau vide
var tableau1 = {};
tableau1["cle1"] = "1er élément";
tableau1["cle2"] = "2eme élément";
// Initialisation d'un tableau avec des éléments
var tableau2 = {
   "cle1": "1er élément",
   "cle2": "2eme élément"
};
```

UQAR

Bases du langage JavaScript (ES6)

→ Parcourir un tableau *associatif*

Afin de parcourir un tableau *associatif* et avoir accès à l'ensemble de ses éléments, on fait appel à une *boucle* **`for`**.

```
var tableau = {
    "cle1": "valeur1",
    "cle2": "valeur2"
 };

 ...

 for(var cle in tableau) {
    alert("Elément pour la clé "+cle+" : "+ tableau[cle]);
 }
```

UQAR

Bases du langage JavaScript (ES6)

→ Les chaînes de caractères

En JavaScript, il est possible de définir une chaîne de caractères par :

```
var chaine1 = "ma chaîne de caractère";
```

Ou par :

```
var chaine1 = new String("ma chaîne de caractère");
```

UQAR

Bases du langage JavaScript (ES6)

→ Méthodes associées aux chaînes de caractères

Méthode	Paramètre	Description
`charAt()`	Index du caractère	*Retourne le caractère localisé à l'index spécifié.*
`concat()`	Chaîne de caractères	*Fusionne deux chaînes de caractères.*
`indexOf()`	Chaîne de caractères	*Retourne l'index la première occurrence.*
`lastIndexOf()`	Chaîne de caractères	*Retourne l'index la dernière occurrence.*
`replace()`	Deux chaînes	*Remplace un bloc par un autre.*
`split()`	Délimiteur	*Découpe une chaîne à l'aide du délimiteur.*
`substr()`	Index de début et fin	*Méthode identique à la méthode* `slice()`.
`toLowerCase()`	-	Convertit en minuscules.
`toUpperCase()`	-	Convertit en majuscules.

Bases du langage JavaScript (ES6)

→ Manipulation des dates

- JavaScript définit les dates uniquement en tant que classe par l'intermédiaire de la classe **`Date`**, laquelle comporte un grand nombre de méthodes.
- La création d'une date se réalise par le biais du *constructeur* de la classe **`Date`**. Ce dernier peut ne prendre aucun paramètre :

```
var dateCourante = new Date();
```

- Une instance de la classe **`Date`** peut également être initialisée à partir d'une date exprimée en millisecondes ou à partir des informations relatives à l'*année*, au *mois*, au *jour* ainsi qu'aux *heures*, aux *minutes*, aux *secondes* et aux *millisecondes*.

```
var uneAutreDate = new Date(1996, 10, 20, 12, 5, 0);
// uneAutreDate correspond au 20 octobre 1996 à 12 heures et 5 minutes
```

Bases du langage JavaScript (ES6)

→ Méthodes associées aux dates

Méthode	Description
`getDate()`	Retourne le jour du mois de la date courante.
`getDay()`	Retourne le jour de la semaine de la date courante.
`getFullYear()`	Retourne l'année de la date courante.
`getHours()`	Retourne les heures de l'heure courante.
`getMilliseconds()`	Retourne les millisecondes de l'heure courante.
`getMinutes()`	Retourne les minutes de l'heure courante.
`getMonth()`	Retourne le mois de la date courante.
`getSeconds()`	Retourne les secondes de l'heure courante.
`getTime()`	Retourne le nombre de la date en millisecondes.

UQAR

Bases du langage JavaScript (ES6)

→ Définition d'une fonction

- Les fonctions représentent le concept de base de la programmation JavaScript afin de *modulariser* les traitements.
- Elles possèdent des spécificités par rapport à d'autres langages tel que JAVA :

```
function nomDeLaFonction(parametre1, parametre2, ...) {
    //Code de la fonction
}
```

- Remarquons que la *définition de la fonction* permet de déterminer son *nom* ainsi que la *liste de ses paramètres*, cette liste ne contenant que leurs noms. Aucune information relative à leur *type* n'est spécifiée. De plus, *le type de retour* n'est pas spécifié, même si le code de la fonction comprend une clause **`return`**.

UQAR

Bases du langage JavaScript (ES6)

→ Appel d'une fonction

Afin d'appeler une fonction en JavaScript, il faut faire comme suit :

```
function test(parametre1, parametre2) {
    return parametre1+" - "+parametre2;
}

var retour = test("param1", "param2");
//La variable retour contient la valeur: "param1 - param2"
```

Bases du langage JavaScript (ES6)

→ Gestion des arguments

- JavaScript met à disposition la liste des arguments passés à une fonction dans une variable particulière,
- Nommée **`arguments`**, cette variable est implicitement définie pour chaque fonction. Le développeur peut donc l'utiliser directement dans le code des fonctions. Cette fonctionnalité permet de supporter plusieurs signatures de méthodes.

```
function maFonction() {
    if( arguments.length == 1 ) {
        return arguments[0];
    }
    if( arguments.length == 2 ) {
        return arguments[0]+" - "+arguments[1];
    }
}
```

Bases du langage JavaScript (ES6)

→ Gestion des *exceptions*

- Comme la plupart des langages objet, JavaScript permet de gérer les exceptions. Ce mécanisme offre la possibilité d'intercepter les erreurs au lieu de les laisser traiter par l'intercepteur du langage.
- Cela évite l'arrêt brutal de l'exécution d'un script et, en conjonction avec un gestionnaire d'exceptions robuste, facilite le débogage des applications.

Bases du langage JavaScript (ES6)

→ Gestion des *exceptions*

- Comme la plupart des langages objet, JavaScript permet de gérer les exceptions. Ce mécanisme offre la possibilité d'intercepter les erreurs au lieu de les laisser traiter par l'intercepteur du langage.
- Cela évite l'arrêt brutal de l'exécution d'un script et, en conjonction avec un gestionnaire d'exceptions robuste, facilite le débogage des applications.
- JavaScript offre la même syntaxe que Java pour gérer les exceptions, au moyen des mots clés **`try`**, **`catch`** et **`finally`**.
- Le premier définit le bloc d'interception des exceptions, le second les traitements à réaliser en cas de levée d'exceptions et le dernier les traitements à exécuter, que des exceptions soient levées ou non.

UQAR

Bases du langage JavaScript (ES6)

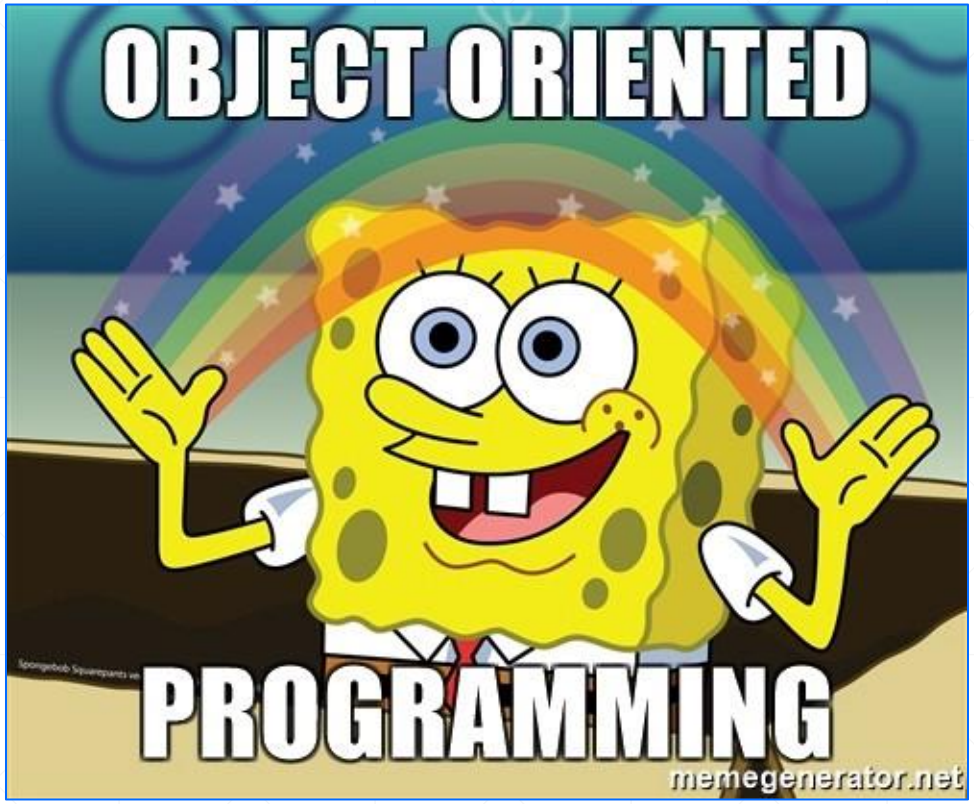
→ Gestion des *exceptions*

Ci-dessous un exemple de gestion d'exceptions :

```
try {
    testException(); //Cette méthode est inexistante!
} catch(error) {
    alert("Une exception a été levée");
    alert("Nom de l'exception levée : " + error.name);
    alert("Message de l'exception levée : " + error.message);
} finally {
    alert("Passage dans finally");
}
```

Bases du langage JavaScript (ES6)

→ Programmation orientée objet



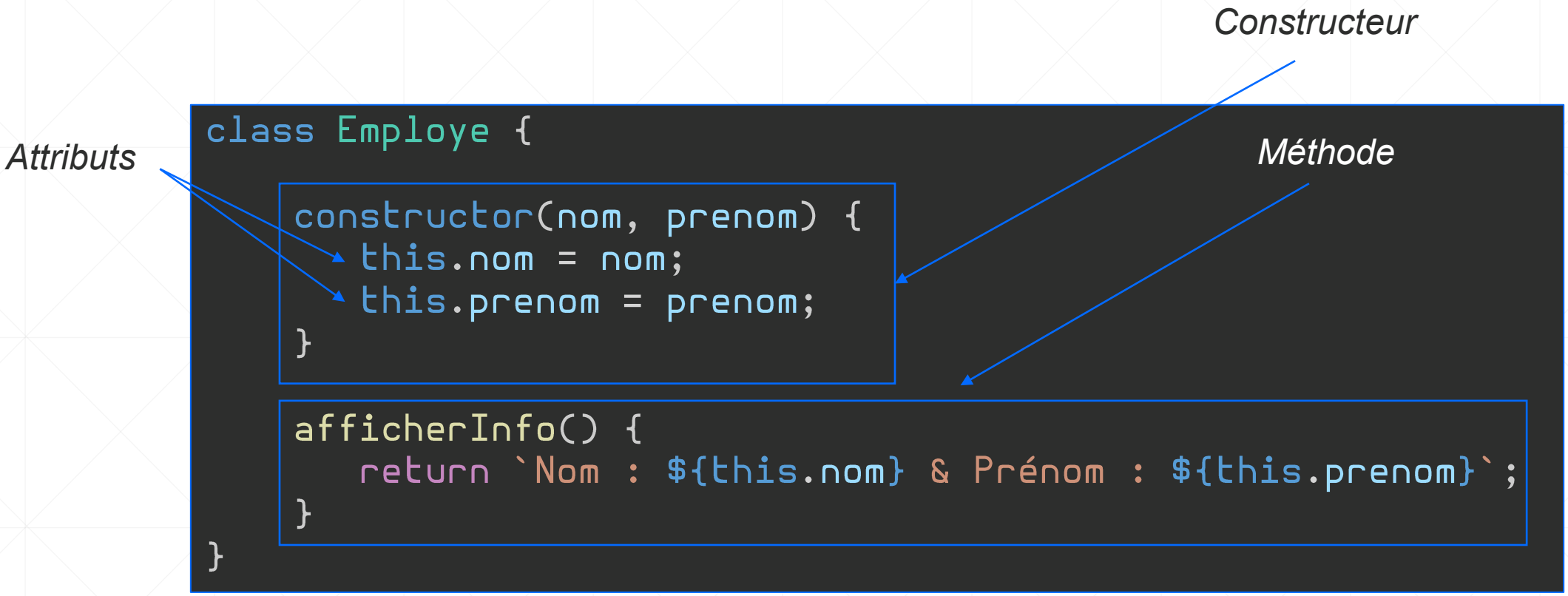
La programmation orientée objet intègre trois principes fondamentaux :

1. **Encapsulation :** *permet de réunir au sein d'une même entité variables (attributs) et fonctions (méthodes).*
2. **Héritage :** *permet de définir une certaine hiérarchie. La classe de base est générique, plus on descend dans la hiérarchie, plus on se spécialise.*
3. **Polymorphisme :** *permet de définir plusieurs fonctions de même nom mais possédant des paramètres différents. L'appel se fera selon les paramètres fournis.*

UQAR

Bases du langage JavaScript (ES6)

→ Création d'une nouvelle *Classe*



UQAR

Bases du langage JavaScript (ES6)

→ Instanciation d'un nouvel *objet*

UQAR

Bases du langage JavaScript (ES6)

→ Héritage & Polymorphisme

Héritage

```
class Directeur extends Employe {
    constructor(nom, prenom, departement) {
        super(nom, prenom);
        this.departement = departement;
    }

    afficherInfo() {
        return `${super.afficherInfo()} | Département : ${this.departement}`;
    }
}
```

Polymorphisme

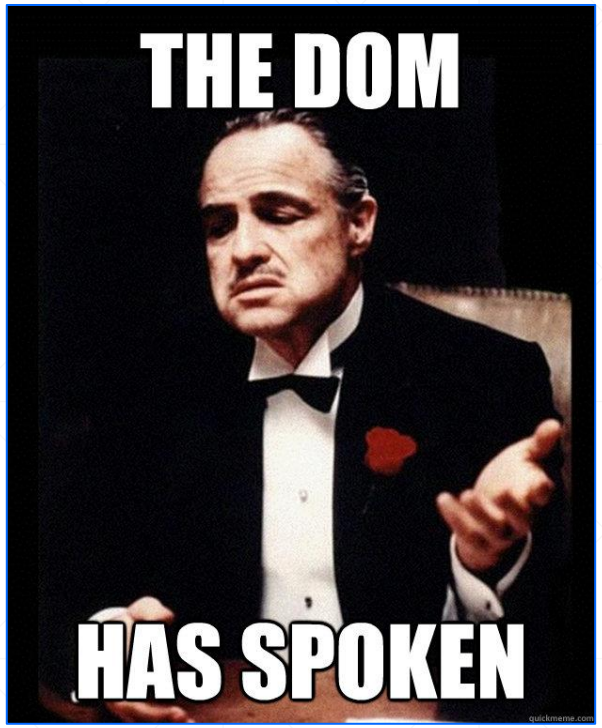
UQAR

Partie IV

Manipulation du DOM et gestion d’événements

Manipulation du DOM et gestion d'événements

→ Le DOM, quésaco ?



- **Définition :** *« **DOM** ou **D**ocument **O**bjet **M**odel est un ensemble d'**interfaces**, normalisée par le **W3C**[1], permettant aux langages de programmation d'accéder aux **objets** qui composent le **document**. »,*
- L'**API**[2] du **DOM** permet d'accéder à une page Web et de manipuler son *contenu*, sa *structure* ainsi que ses *styles*,
- Il fournit une représentation objet normalisée des documents, dont le contenu est **arborescent** : pages **HTML** ou documents **XML**.

[1] ***W**orld **W**ide **W**eb **C**onsortium*
[2] ***A**pplication **P**rogramming **I**nterface*

UQAR

Manipulation du DOM et gestion d'événements

→ Un peu plus de détails

*Le **DOM** définit la manière dont les navigateurs Web créent un modèle d'une page HTML et comment le JavaScript peut accéder et mettre à jour le contenu de la page Web alors qu'elle est affichée au niveau de la fenêtre du navigateur Web.*

Modélisation de pages **HTML**

- Lors de son chargement, le navigateur Web crée un modèle en mémoire de la page Web,
- Le **DOM** définit la structure du modèle en utilisant une arborescence spécifique,
- Elle constituée d'objet dont chacune décrit une partie spécifique de la page.

Accès & modification de pages **HTML**

- Le **DOM** définit un ensemble de *méthodes* et de *propriétés* afin d'*accéder* et de *modifier* chacun des objets du modèle,
- On peut faire référence au **DOM** par **API**. Elle permet au script de récupérer à partir du navigateur des informations sur la page Web affichée.
- Elle permet aussi au script de demander au navigateur de faire des modifications sur la page affichée.

UQAR

Manipulation du DOM et gestion d'événements

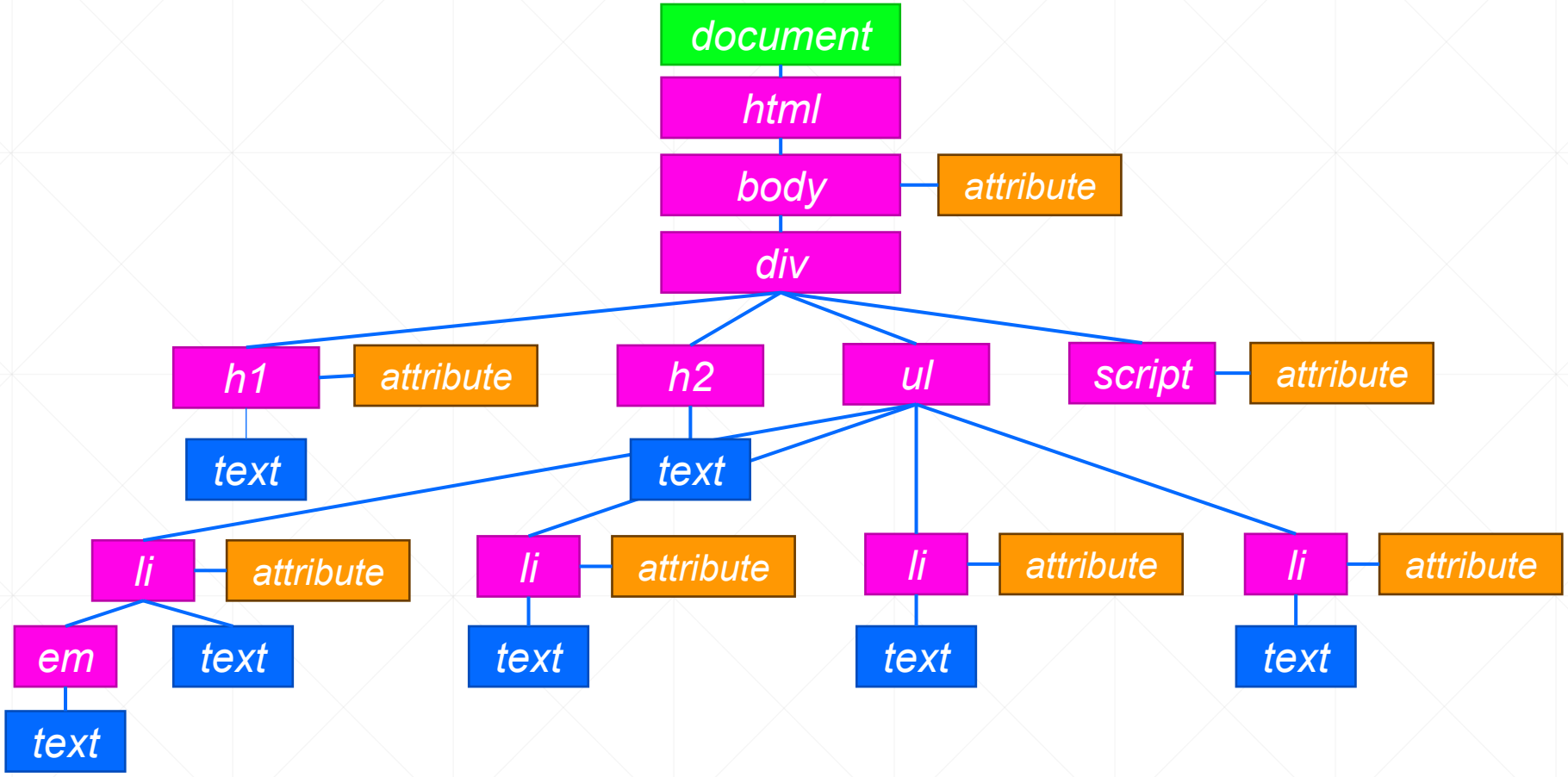
→ Modèle par arborescence

```
<!DOCTYPE html>

<html>
    <head>
        <meta charset="utf-8" />
        <title></title>
    </head>
    <body>
        <div id="page">
            <h1 id="header">Liste</h1>
            <h2>Faire les courses</h2>
            <ul>
                <li id="one" class="hot">Figues <br><em>fraîches</em></li>
                <li id="two" class="hot">Beurre d'arachide</li>
                <li id="three" class="hot">Avocats</li>
                <li id="four">Bananes</li>
            </ul>

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

UQAR

Manipulation du DOM et gestion d'événements

→ Représentation structurée



Manipulation du DOM et gestion d'événements

→ Les différents nœuds

Nœud Document

- Il se trouve au tout début de l'arborescence du **DOM**,
- Il représente la page Web entière,
- Il correspond à `document object`,
- L'accès à n'importe quel nœud élément, attributs ou texte doit se faire à travers le nœud `document`.

Nœuds Eléments

- Les éléments HTML décrivent la structure d'une page Web,
- Afin d'accéder à l'arborescence du **DOM**, on doit commencer par localiser l'élément ciblé,
- La première étape est toujours l'accès aux éléments avant celui des attributs et du texte.

UQAR

Manipulation du DOM et gestion d'événements

→ Les différents nœuds

Nœuds Attributs

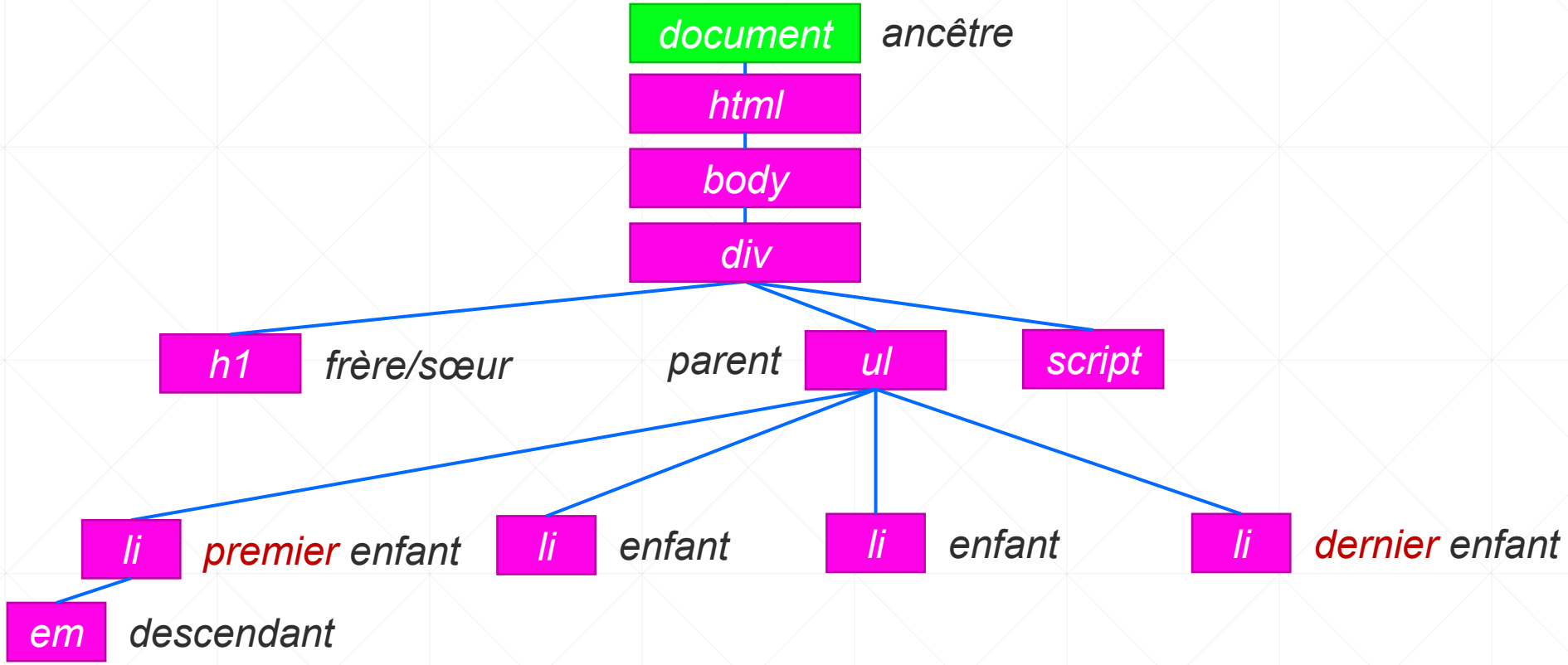
- Au sein des éléments HTML, on peut y disposer (à l'intérieur des **tags**) des attributs ayant des valeurs,
- Les attributs font partie intégrante de l'élément HTML,
- Il y a des méthodes et propriétés en JavaScript permettant de modifier les valeurs des attributs.

Nœuds Texte

- Une fois avoir accédé au nœud élément, il est possible d'atteindre le texte qu'il contient,
- Les nœuds texte représente les derniers éléments de l'arborescence **DOM**,
- Par conséquent, ils ne peuvent pas avoir d'enfants.

Manipulation du DOM et gestion d'événements

→ Relations & Liens entre éléments



UQAR

Manipulation du DOM et gestion d'événements

→ Accès aux éléments

Il y a trois types de méthodes et propriétés permettant d'accéder aux différents éléments de l'arborescence du **DOM** :

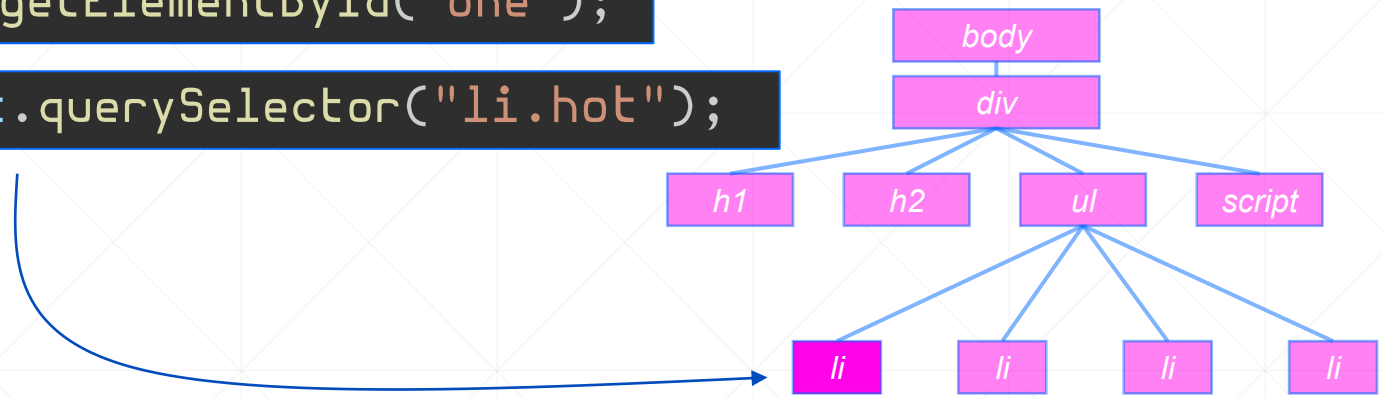
- **Accès individuel :** *permet d'avoir un accès à un élément en spécifique ou déterminé, il n'y a pas de confusion possible.*
- **Accès multiple :** *permet d'avoir accès à une liste d'éléments ayant les des caractéristiques communes.*
- **Accès par relation :** *permet de naviguer ou de se déplacer d'un élément à un autre en utilisant un critère relationnel : parent, enfant, frère/sœur.*

Manipulation du DOM et gestion d'événements

→ Accès *individuel*

```
var itemOne = document.getElementById("one");
```

```
var itemOne = document.querySelector("li.hot");
```

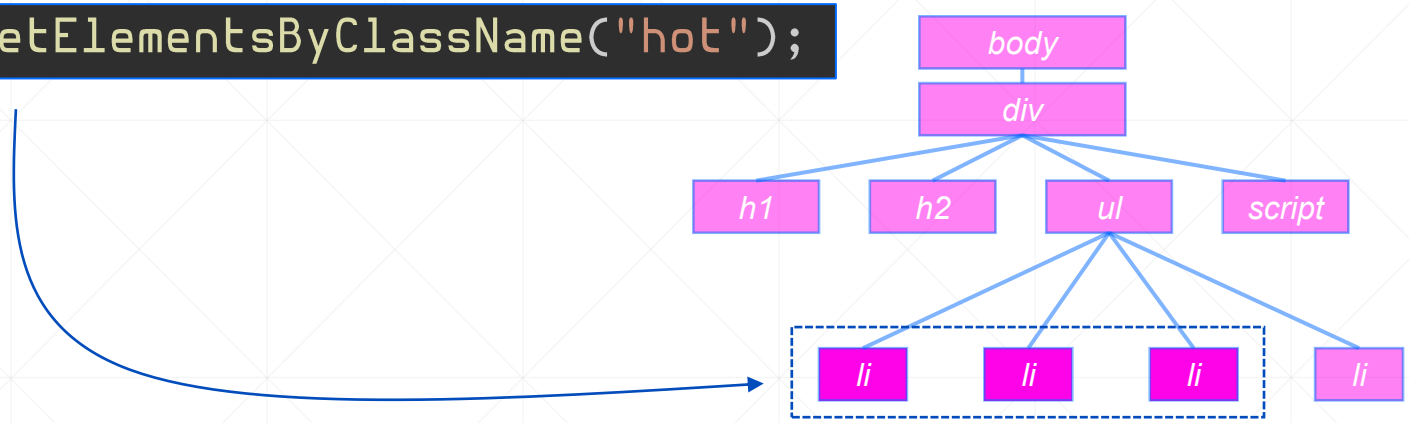


Si l'on souhaite utiliser un élément particulier plusieurs fois dans un script JavaScript, il est possible de *stocker sa position* dans le **DOM** à travers une *variable*.

Manipulation du DOM et gestion d'événements

→ Accès multiple

```
var items = document.getElementsByClassName("hot");
```



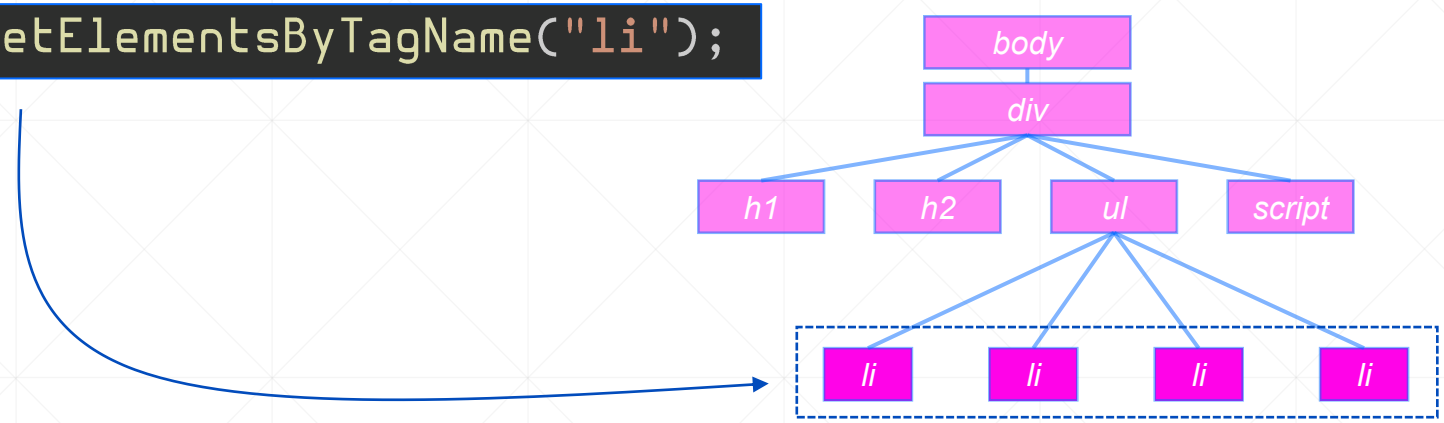
Elle permet de sélectionner une *liste d'éléments* en utilisant le nom de la **classe** qui leurs est attribuée.

UQAR

Manipulation du DOM et gestion d'événements

→ Accès multiple

```
var items = document.getElementsByTagName("li");
```



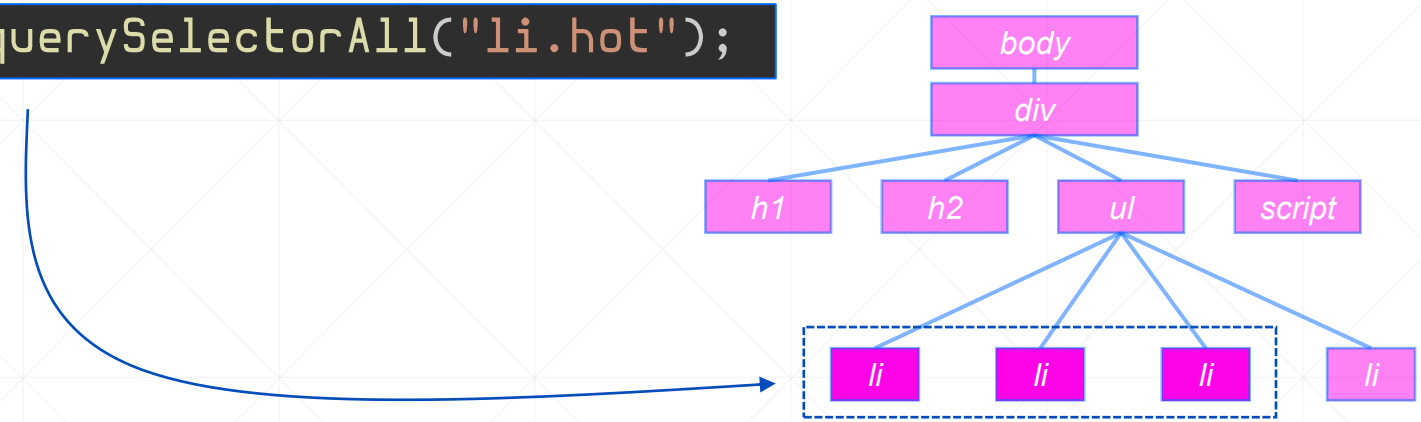
Elle permet de sélectionner *tous les éléments* de la page HTML ayant un **tag** ou un **type de balise** spécifique.

UQAR

Manipulation du DOM et gestion d'événements

→ Accès multiple

```
var items = document.querySelectorAll("li.hot");
```



Permet de retourner *tous les éléments* de la page HTML correspondant aux **sélecteur CSS** utilisé.

UQAR

Manipulation du DOM et gestion d'événements

→ Manipuler une liste de nœuds élément

Afin de travailler avec une liste de nœuds élément, il y a deux méthodes :

```
var items = document.querySelectorAll("li.hot");
```

```
if (items.length >= 1) {
    var firstItem = items.item(0);
}
```

Ou

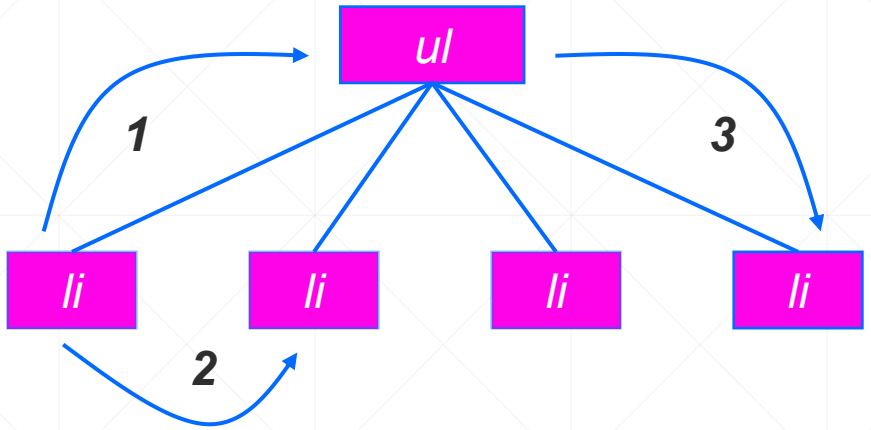
```
if (items.length >= 1) {
    var firstItem = items[0];
}
```

Manipulation du DOM et gestion d'événements

→ Accès par relation

Il y a cinq types d'accesseur par relation :

- `parentNode`
- `previousSibling`
- `nextSibling`
- `firstChild`
- `lastChild`



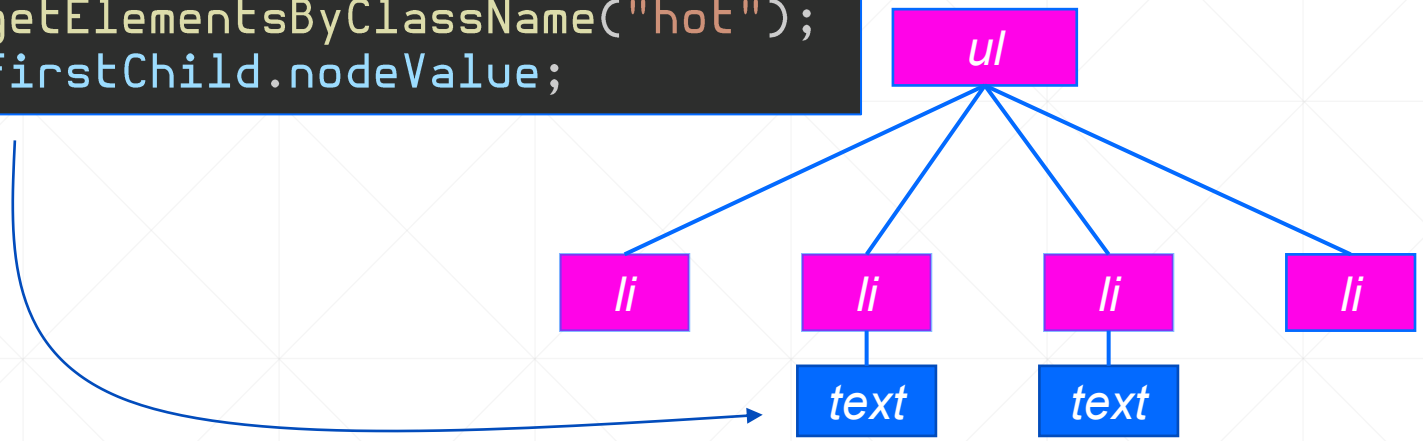
1. Fait référence au **nœud parent**,
2. Fait référence au **premier nœud frère/sœur**,
3. Fait référence au **dernier nœud enfant**.

UQAR

Manipulation du DOM et gestion d'événements

→ Manipuler les nœuds texte

```
var items = document.getElementsByClassName("hot");
var value = items[1].firstChild.nodeValue;
```



L'attribut **`nodeValue`** permet d'avoir accès en lecture et en écriture à la valeur du nœud texte d'un certain nœud élément.

UQAR

Manipulation du DOM et gestion d'événements

→ Manipuler le texte dans les nœuds élément

Il y a deux principales méthodes afin de *récupérer* et *manipuler* le contenu texte :

- **`textContent` :** *cet attribut permet de récupérer le contenu textuel d'un nœud en particulier ainsi que celui de ses descendants.*

```
var items = document.getElementsByClassName("hot");
var text = items[0].textContent;
```

→ `"Figues fraîches"`

- **`innerText` :** *cet attribut permet de récupérer le contenu textuel « visuellement rendu » d'un nœud en particulier.*

```
var items = document.getElementsByClassName("hot");
var text = items[0].innerText;
```

→ `"Figues fraîche"`

UQAR

Manipulation du DOM et gestion d'événements

→ Manipuler les balises & le texte

L'attribut **`innerHTML`** permet de récupérer et modifier le contenu d'un nœud élément comprenant le *texte* ainsi que les *balises*.

- Récupérer du contenu :

```
var content = document.getElementsById("one").innerHTML;
```

- Mettre à jour le contenu :

```
document.getElementsById("one").innerHTML = content;
```

Manipulation du DOM et gestion d'événements

→ Manipuler les balises & le texte

L'attribut **`outerHTML`** est similaire à **`innerHTML`**. La principale différence réside dans le fait qu'elle ne récupère pas uniquement le contenu des enfants mais de la balise ciblée également.

- Récupérer du contenu :

```
var content = document.getElementsById("one").outerHTML;
```

- Mettre à jour le contenu :

```
document.getElementsById("one").outerHTML = content;
```

UQAR

Manipulation du DOM et gestion d'événements

→ Ajout d'un élément au DOM

Cette opération se déroule suivant trois étapes principales :

1. Création de l'élément :

```
var newElement = document.createElement("li");
```

2. Affectation du contenu :

```
var newText = document.createTextNode("Jus d'orange");
newElement.appendChild(newText);
```

3. Ajout au **DOM** :

```
var position = document.getElementsByTagName("ul")[0];
position.appendChild(newElement);
```

UQAR

Manipulation du DOM et gestion d'événements

→ Suppression d'un élément du DOM

Cette opération se déroule suivant trois étapes principales :

1. Récupérer l'élément du **DOM** à supprimer :

```
var rmElement = document.getElementsByTagName("li")[3];
```

2. Récupérer l'élément parent :

```
var parentElement = rmElement.parentNode;
```

3. Supprimer l'élément du parent :

```
parentElement.removeChild(rmElement);
```

UQAR

Manipulation du DOM et gestion d'événements

→ Vérifier l'existence & récupération d'un *attribut*

- Afin de vérifier l'existence d'un attribut associé à un élément :

```
var firstItem = document.getElementById("one");

if (firstItem.hasAttribute("class")) {
    ...
}
```

- Afin de récupérer sa valeur :

```
var attr = firstItem.getAttribute("class");
```

UQAR

Manipulation du DOM et gestion d'événements

→ Créer & changer la valeur d'un *attribut*

- Afin de créer un nouvel attribut :

```
var fourthItem = document.getElementsByTagName("li").item(3);
fourthItem.setAttribute("class", "cool");
```

- Afin de changer la valeur d'un attribut existant :

```
var firstItem = document.getElementById("one");
firstItem.className = "complete";
```

UQAR

Manipulation du DOM et gestion d'événements

→ Suppression d'un *attribut*

Afin de supprimer un attribut à partir d'un élément donné, il faut :

```
var firstItem = document.getElementById("one");
if (firstItem.hasAttribute("class")) {
   firstItem.removeAttribute("class");
}
```

UQAR

Manipulation du DOM et gestion d'événements

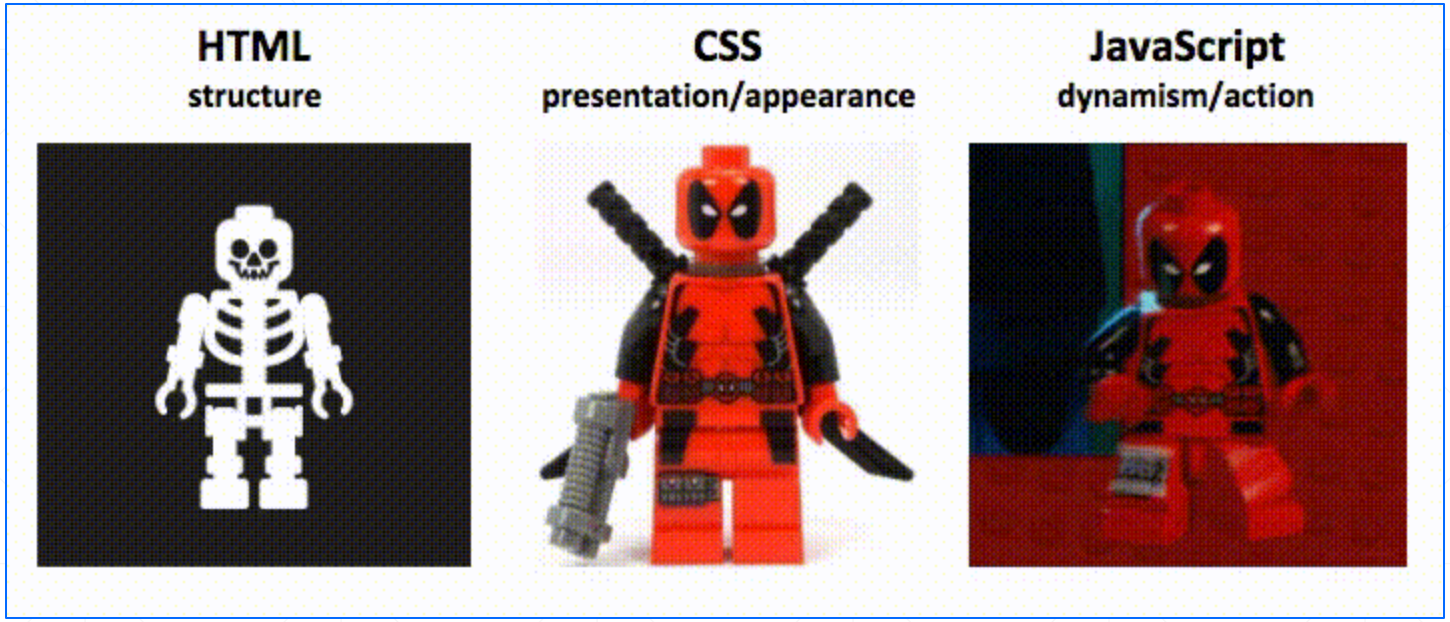
→ Changer le styles CSS

Afin de pouvoir modifier une des propriétés **CSS** d'un élément du **DOM** :

```
var firstItem = document.getElementById("one");
firstItem.style.color = "blue";
```

Manipulation du DOM et gestion d'événements

→ Interagir avec l'utilisateur



UQAR

Manipulation du DOM et gestion d'événements

→ Événements, quésaco ?

- **Définition :** *« ce sont des actions qui se produisent et auxquelles on va pouvoir répondre en exécutant du code. »,*
- Les événements et leur prise en charge est l'un des mécanismes fondamentaux du JavaScript qui vont nous permettre d'ajouter du dynamisme et de l'interactivité aux pages Web.
- Un événement en JavaScript possède deux caractéristiques :
  1. **Ecouter :** *une action peut être détectée car le système (le navigateur Web) va nous informer qu'elle se produit.*
  2. **Répondre :** *on a la possibilité d'attacher un code à cette action qui va s'exécuter dès qu'elle va être détectée.*

Manipulation du DOM et gestion d'événements

→ Type d'événements

Il y a toutes sorte d'événements qu'il est possible d'utiliser dans une page Web, il y a deux catégories :

1. Des événements qui s'appliquent sur tous les éléments du **DOM** ou de la page Web en général,
2. Des événements qui s'appliques uniquement sur les *formulaires* ou bien pour être plus précis, sur la balise :

```
<form action="/" method="post"></form>
```

UQAR

Manipulation du DOM et gestion d'événements

→ Liste des événements I

événement	Action à déclencher
`click`	*Cliquer sur un élément (appuyer et relâcher)*
`dbclick`	*Double-cliquer sur un élément*
`mouseover`	*Faire entrer le curseur sur un élément*
`mouseout`	*Faire sortir le curseur de l'élément*
`keypress`	*Cliquer sur une touche du clavier sur un élément*
`change`	*Changer la valeur d'un élément spécifique aux formulaires*
`input`	*Taper un caractère sur un champs de texte*
`select`	*Sélectionner le contenu dans un champs de texte*

UQAR

Manipulation du DOM et gestion d'événements

→ Liste des événements II

Les événements ci-dessous s'appliquent exclusivement aux formulaires :

événement	Action à déclencher
`submit`	*Envoyer le formulaire au serveur*
`reset`	*Réinitialiser les différents champs du formulaire*

UQAR

Manipulation du DOM et gestion d'événements

→ Associer un événement

- Afin d'ajouter ou bien d'associer un événement à un élément précis de la page Web, on doit faire comme suit :

```
<button type="button" onclick="alert('Vous avez cliqué !');">Cliquez-moi !</button>
```

- C'est l'attribut **`onclick`** qui se charge d'ajouter un événement et sa valeur représente l'action à exécuter.

```
<input value="Cliquez-moi !" onclick="this.value = 'Vous avez cliqué !';"></input>
```

- Le mot clé **`this`** renvoie à l'élément lui-même, il est donc possible d'apporter de modification à ce dernier une fois l'événement déclenché.
- Il est possible de bloquer l'exécution d'un événement en ajouter `return false;`

UQAR

Manipulation du DOM et gestion d'événements

→ À partir d'un script

- Il est possible d'associer des événements à des éléments du **DOM** directement à partir de scripts, pour cela, il faut faire comme suit :

```
var element = document.getElementById("myButton");

element.onclick = function () {
   alert("Vous m'avez cliqué !");
};
```

- Il suffit de récupérer l'élément cible du **DOM**, ensuite lui associer une *fonction* qui représente l'action à exécuter.

Manipulation du DOM et gestion d'événements

→ Plusieurs événements

Il est possible d'associer plusieurs actions à un même type d'événement, il faut faire comme suit :

```
var element = document.getElementById("myButton");

element.addEventListener('click', function() {
   alert("Et de un !");
});

var myFunction = function() {
   alert("Et de deux !");
};

element.addEventListener('click', myFunction);
```

UQAR

Manipulation du DOM et gestion d'événements

→ Suppression d'un événement

Il est possible d'enlever ou de *supprimer* un événement parmi ceux qui ont été associé à un élément du **DOM** en faisant :

```
var element = document.getElementById("myButton");

element.addEventListener('click', function() {
   alert("Et de un !");
});

var myFunction = function() {
   alert("Et de deux !");
};

element.addEventListener('click', myFunction);
element.removeEventListener('click', myFunction);
```

Manipulation du DOM et gestion d'événements

→ Ordre de déclenchement

Dans la fonction **`addEventListner()`**, il est possible d'ajouter un troisième argument de type **`boolean`** qui définit l'ordre de déclenchement des événements :

1. **Mode *capture* (`true`) :** L'événement est capturé en partant des éléments parents et se propage vers les éléments enfants. Cela signifie que l'écouteur d'événements sera exécuté d'abord sur l'élément parent le plus élevé dans la hiérarchie du DOM, avant de descendre vers l'élément enfant ciblé.

2. **Mode *bouillonnement* (`false`) :** L'événement est d'abord capturé par l'élément enfant où l'événement a eu lieu, puis il « *remonte* » dans la hiérarchie du DOM vers les parents. C'est la phase par défaut dans JavaScript.

UQAR

Questions & Discussion

UQAR

Bibliographie

1. Templier, Thierry & Gougeon, Arnaud (2007). JavaScript pour le Web 2.0 Programmation objet, DOM, Ajax, Prototype, Dojo, Script.aculo.us, Rialto. Éditions Eyrolles.
2. Porteneuve, Christophe (2008). Bien développer pour le Web 2.0 : Bonnes pratiques Ajax. Éditions Eyrolles.
3. Engels, Jean (2012). HTML5 et CSS3 : Cours et exercices corrigés. Éditions Eyrolles.
4. Martin, Michel (2014). HTML5, CSS3 & jQuery : Créez votre premier site web. Éditions Pearson.