Technologies Web
JavaScript/jQuery

2021-2022

Plan

Introduction (1/5)

Généralités

- JavaScript
  - Créé en 1995 par Netscape et Sun Microsystems
  - Fait partie des langages web dits « standards » avec le HTML et le CSS
  - Le JavaScript est un langage dynamique, un langage (principalement) côté client, un langage interprété, un langage orienté objet,
  - But: interactivité dans les pages HTML, traitements simples sur le poste de travail de l'utilisateur
  - Moyen : introduction de scripts dans les pages HTML
  - Norme: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/

Ce langage va nous permettre de manipuler des contenus HTML ou des styles CSS et de les modifier en fonction de divers évènements ou variables.
Un évènement peut être par exemple un clic d'un utilisateur à un certain endroit de la page tandis qu'une variable peut être l'heure de la journée.

Introduction (2/5)

Avantages et inconvénients

- Points forts :
  - langage de programmation structurée ; de nombreuses applications sont maintenant développées uniquement en JavaScript, côté serveur (en utilisant par exemple Node.js)
  - il enrichit le HTML (intégré) interprété par le client),
  - il partage les prototypes DOM des documents HTML/XHTML) => manipulation dynamique possible,
  - gestionnaire d'événements (programmation asynchrone possible)

- Limitations/dangers :
  - c'est un langage de script (interprété), très permissif
  - typage faible

Il est recommandé de bien suivre les bonnes pratiques !

Introduction (3/5)

Domaines d'application

- JavaScript permet:
  - de programmer des actions en fonction d'événements utilisateurs (déplacements de souris, focus, etc.) ;
  - d'accéder aux éléments de la page HTML (traitement de formulaire, modification de la page)
  - d'effectuer des calculs sans recours au serveur

- Domaines d'application historiques :
  - petites applications simples (calculette, conversion, etc.)
  - aspects graphiques de l'interface (événements, fenêtrage, etc.)
  - tests de validité sur des interfaces de saisie

- Exemples de nouvelles applications possibles en JavaScript :
  - Vidéos affichées en HTML5 sans Flash (ex : YouTube)
  - Jeux
  - Bureautique (ex : Google Docs)

Introduction (4/5)

Normalisation

- ,
  - ECMA (European Computer Manufactures Association) a défini un standard ECMAScript (Mozilla et Adobe);
  - JavaScript 1.8.5: http://www.ecma-international.org/publications/standards/Ecma-262.htm
  - Ce standard, repris par l'ISO, définit les caractéristiques du noyau du langage
  - JavaScript 2.0 est conforme à cette norme mais a ses propres extensions et des différences au niveau du modèle objet du navigateur

Introduction (5/5)

jQuery

- Description,
  - Framework JavaScript (comme PrototypeJS, Mootools, Dojo, YahooUI, ...);
  - Créé en Janvier 2006
  - Très utilisé : >50% des sites dans le monde
  - 3 versions : 1.X (1.11.3), 2.X (2.1.4), 3.X (3.6.0)

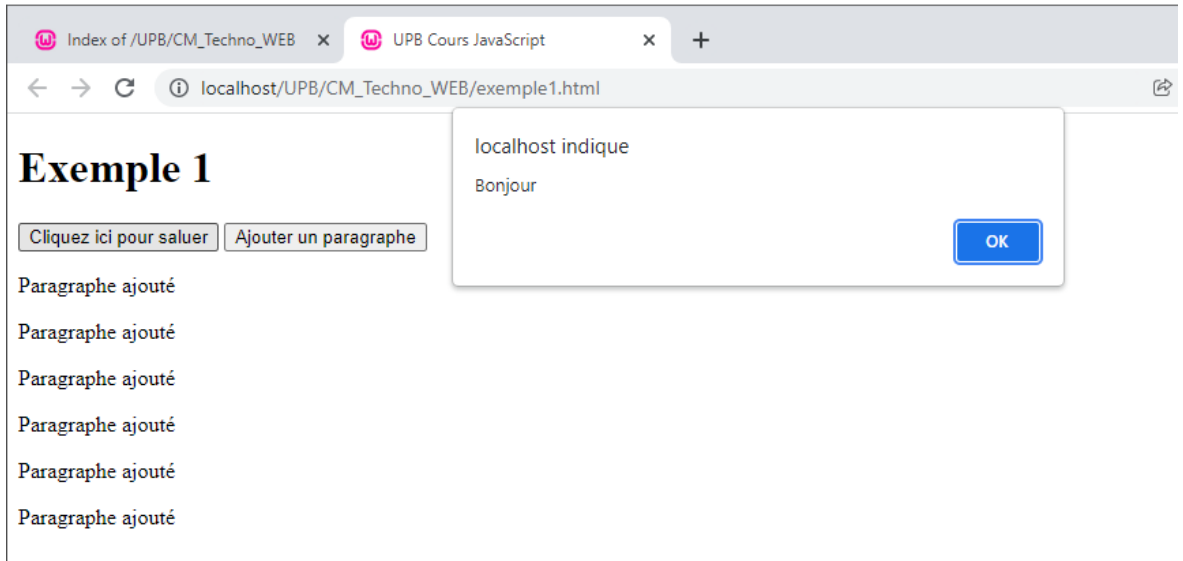
- Objectifs :
  - Régler les problèmes de compatibilité entre navigateurs
  - Faciliter l'écriture de scripts

Inclusion de code JavaScript/jQuery (1/6)

Où écrire le code JavaScript ?

- On va pouvoir placer du code JavaScript à trois endroits différents :
  - Directement dans la balise ouvrante d'un élément HTML ;
  - Dans un élément script, au sein d'une page HTML ;
  - Dans un fichier séparé contenant exclusivement du JavaScript et portant l'extension js

- Exemple:



Inclusion de code JavaScript/jQuery (2/6)

Où écrire le code JavaScript ?

- Placer le code JavaScript dans la balise ouvrante d'un élément HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>UPB Cours JavaScript</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Exemple 1</h1>
        <button onclick="alert('Bonjour')"> Cliquez ici pour saluer</button>

        <button onclick="(function(){
            let para = document.createElement('p');
            para.textContent = 'Paragraphe ajouté';
            document.body.appendChild(para);
        })();">
            Ajouter un paragraphe
        </button>
    </body>
</html>
```

Inclusion de code JavaScript/jQuery (3/6)

Où écrire le code JavaScript ?

- Placer le code JavaScript dans un élément script, au sein d'une page HTML

Ce genre de syntaxe est généralement déconseillé et considéré comme une mauvaise pratique

```
<!DOCTYPE html>
<html>
    <head>
        <title>UPB Cours JavaScript</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
        <script>
            document.addEventListener('DOMContentLoaded', function(){
                let bonjour = document.getElementById('b1');
                bonjour.addEventListener('click', alerte);

                function alerte(){
                    alert('Bonjour');
                }
            });
        </script>
    </head>

    <body>
        <h1>Titre principal</h1>
        <button id='b1'>Cliquez ici pour saluer</button>
        <button id='b2'>Ajouter un paragraphe</button>
        <script>
            let ajouter = document.getElementById('b2');
            ajouter.addEventListener('click', ajout);
            function ajout(){
                let para = document.createElement('p');
                para.textContent = 'Paragraphe ajouté';
                document.body.appendChild(para);
            }
        </script>
    </body>
</html>
```

Inclusion de code JavaScript/jQuery (4/6)

Où écrire le code JavaScript ?

- Placer le code JavaScript dans un élément script, au sein d'une page HTML

Cette méthode est meilleure que la précédente mais le fait qu'on mélange du JavaScript et du HTML peut rendre l'ensemble confus et complexe à comprendre dans le cadre d'un gros projet.

```
<!DOCTYPE html>
<html>
    <head>
        <title>UPB Cours JavaScript</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
        <script>
            document.addEventListener('DOMContentLoaded', function(){
                let bonjour = document.getElementById('b1');
                bonjour.addEventListener('click', alerte);

                function alerte(){
                    alert('Bonjour');
                }
            });
        </script>
    </head>

    <body>
        <h1>Titre principal</h1>
        <button id='b1'>Cliquez ici pour saluer</button>
        <button id='b2'>Ajouter un paragraphe</button>
        <script>
            let ajouter = document.getElementById('b2');
            ajouter.addEventListener('click', ajout);
            function ajout(){
                let para = document.createElement('p');
                para.textContent = 'Paragraphe ajouté';
                document.body.appendChild(para);
            }
        </script>
    </body>
</html>
```

Inclusion de code JavaScript/jQuery (5/6)

Où écrire le code JavaScript ?

- Placer le code JavaScript dans un fichier séparé

exemple1_c.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours JavaScript</title>
        <meta charset="utf-8">
        <meta name="viewport"
          content="width=device-width, initial-scale=1, user-scalable=no">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <button id='b1'>Cliquez moi</button>
        <button id='b2'>Ajouter un paragraphe</button>

        <script type="text/javascript" src="cours.js"></script>
    </body>
</html>
```

cours.js

```
let bonjour = document.getElementById('b1');
let ajouter = document.getElementById('b2');

bonjour.addEventListener('click', alerte);
ajouter.addEventListener('click', ajout);

function alerte(){
    alert('Bonjour');
}
function ajout(){
    let para = document.createElement('p');
    para.textContent = 'Paragraphe ajouté';
    document.body.appendChild(para);
}
```

C’est la méthode conseillée, elle permet une excellente séparation du code et une maintenabilité optimale de celui-ci

Inclusion de code JavaScript/jQuery (6/6)

Inclusion du bibliothèque jQuery

- Utilisation d'une version locale
  - Téléchargement sur www.jquery.com
  - <script type="text/ javascript " src="jquery. js "></script>

- Utilisation de version en cache
  - Un réseau de diffusion de contenu (RDC) ou en anglais **content delivery network (CDN)** est constitué d'ordinateurs reliés en réseau à travers Internet et qui coopèrent afin de mettre à disposition du contenu ou des données à des utilisateurs.
  - Google CDN :
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
  - Microsoft CDN :
    <script src=" https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.6.0.js"></script>
  - CDNJS CDN :
  - jsDelivr CDN :

Syntaxe générale(1/6)

Caractéristiques

- Description de la syntaxe,
  - Variables faiblement typées
  - Opérateurs et instructions identiques au C/C++/Java
  - Des fonctions/procédures
    - globales (méthodes associées à tous les objets)
    - fonctions/procédures/méthodes définies par l'utilisateur
  - Des objets (des prototypes)
    - prédéfinis (String, Date, Math, etc.)
    - liés à l'environnement
    - définis par l'utilisateur
  - Commentaires : // ou /*...*/
  - Séparateur d'instruction : ';'

Syntaxe générale(2/6)

Opérateurs

- Opérateurs identiques à ceux du C/C++/Java ,
  - opérateurs arithmétiques : + , - , * , / , %
  - in/décrémentation : var++ , var-- , ++var , --var
  - opérateurs logiques : && , | | , !
  - comparaisons : == , === , != , !== , <= , < , >= , >
  - concaténation de chaîne de caractères : +
  - affectation : = , += , -= , *= ...

Opérateur (nom)	Opérateur (symbole)	Description
AND (ET)	&&	Lorsqu'il est utilisé avec des valeurs booléennes, renvoie true si toutes les comparaisons sont évaluées à true ou false sinon
OR (OU)	||	Lorsqu'il est utilisé avec des valeurs booléennes, renvoie true si au moins l'une des comparaisons est évaluée à true ou false sinon
NO (NON)	!	Renvoie false si une comparaison est évaluée à true ou renvoie true dans le cas contraire

Opérateur	Définition
==	Permet de tester l'égalité sur les valeurs
===	Permet de tester l'égalité en termes de valeurs et de types
!=	Permet de tester la différence en valeurs
<>	Permet également de tester la différence en valeurs
!==	Permet de tester la différence en valeurs ou en types
<	Permet de tester si une valeur est strictement inférieure à une autre
>	Permet de tester si une valeur est strictement supérieure à une autre
<=	Permet de tester si une valeur est inférieure ou égale à une autre
>=	Permet de tester si une valeur est supérieure ou égale à une autre

Syntaxe générale(3/6)

Variables

❑ Utilisation de variables,

- Une variable est un conteneur servant à stocker des informations de manière temporaire, comme une chaine de caractères (un texte) ou un nombre par exemple.
  - le nom des variables est sensible à la casse (texte, TEXTE et tEXTe => totalement différent)
  - Distinction de la localisation des variables (locale ou globale –déclarée en dehors d'une fonction-)
  - Typage dynamique (à l'affectation)
- Déclaration : var nom[=valeur] ou let nom[=valeur] ;
  - Exemple: var monAge ou let monAge

Syntaxe générale(3/6)

Variables

- Utilisation de variables,

```
/****** Déclaration et initialisation des variables ***/
// On déclare et on initialise la variable en même temps
let prenom = 'Eric';

// On déclare la variable puis on l'initialise ensuite
let monAge;
monAge = 20;

/*On modifie la valeur stockée dans prenom.
*Notre variable stocke désormais la valeur "Myriam"*/
prenom = 'Myriam';


/***** les variables utilisant la syntaxe let doivent obligatoirement
être déclarées avant de pouvoir être utilisées. ****/

//Ceci fonctionne
prenom = 'Paulin';
var prenom;

//Ceci ne fonctionne pas et renvoie une erreur
nom = 'Arthur';
let nom;


/**** La nouvelle syntaxe avec let n'autorise pas  ****/
//Ceci fonctionne
var prenom = 'Paulin';
var prenom = "Myriam";

//Ceci ne fonctionne pas et renvoie une erreur
let nom = "Gerard";
let nom = 'Akissi';
```

Syntaxe générale(4/6)

Tests et boucles

❑ Si-sinon-alors,

```
if ( condition ) {
    instructions
}
[ else if (condition ) {
    instructions
} ]
[ else {
    instructions
} ]
```

❑ Switch-case,

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

❑ Boucles for,

```
for ( i=0 ; i<N ; i++) {
  instructions
}

for ( p in tableau ) {
 instructions
}
```

❑ Boucles while,

```
while ( condition ) {
  instructions
}

do{
   instructions
} while (condition ) ;
```

Syntaxe générale(5/6)

Fonctions/Procédures

- Les fonctions sont des blocs de code nommés et réutilisables et dont le but est d'effectuer une tâche précise ;
- Il existe deux grands types de fonctions en JavaScript : les fonction natives ou prédéfinies (qui sont en fait des méthodes) qu'on n'aura qu'à appeler et les fonctions personnalisées qu'on va pouvoir créer ;
- On crée une fonction personnalisée grâce au mot clef function

```
function nom (arg1 , ... , argN ) {
   Instructions
[ return valeur ;]
}
```

Syntaxe générale(6/6)

Les tableaux

- En JavaScript, les tableaux sont avant tout des objets qui dépendent de l'objet global Array;
- Déclaration:
  - var nom = new Array ([dimens i on]) ; var nom = new Ar ray (o1 , . . . , on) ;
  - syntaxe utilise les crochets [] : let fruits = [];

```
//Pour créer un tableau vide et l'enregistrer dans une variable, utilisez une paire de crochets :
let fruits = [];

// Vous pouvez aussi créer un tableau rempli en plaçant les éléments voulus à l'intérieur de ces crochets :
let guests = ["Mangue", "pomme", "banane"];
```

- Accession avec [ ] (ex : tableau [i])
  - les indices varient de 0 à N-1
  - les éléments peuvent être de type différent
  - la taille peut changer dynamiquement
  - les tableaux à plusieurs dimensions sont possibles

Syntaxe générale(6/6)

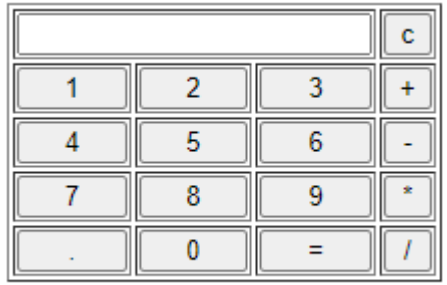
Les tableaux

- Propriétés et méthodes;
  - Le constructeur Array() ne possède que deux propriétés : la propriété length qui retourne le nombre d'éléments d'un tableau et la propriété prototype qui est une propriété que possèdent tous les constructeurs en JavaScript
  - Array() possède une trentaine de méthodes dont:
    - Les méthodes push() et pop() : Ajouter en fin de tableau / supprimer le dernier élément
    - Les méthodes unshift() et shift() : Ajouter au début/ supprimer le 1er élément
    - La méthode splice(): permet d'ajouter, de supprimer ou de remplacer des éléments n'importe où dans un tableau
    - La méthode join() : retourne une chaine de caractères créée en concaténant les différentes valeurs d'un tableau, séparateur (,)
    - La méthode slice() : renvoie un tableau créé en découpant un tableau de départ
    - La méthode concat() : va nous permettre de fusionner différents tableaux entre eux pour en créer un nouveau qu'elle va renvoyer,
    - La méthode includes(): permet de déterminer si un tableau contient une valeur qu'on va passer en argument, renvoie true/false

Syntaxe générale(6/6)

Exemples

- Calculatrice basique



Les prototypes / L'Orienté Objet en JavaScript (1/12)

Paradigmes de programmation

- Il existe trois paradigmes de programmation particulièrement populaires, c'est-à-dire trois grandes façons de penser son code :
  - La programmation procédurale ;
  - La programmation fonctionnelle ;
  - La programmation orientée objet.

Chacun de ces paradigmes ne correspond qu'à une façon différente de penser, d'envisager et d'organiser son code et qui va donc obéir à des règles et posséder des structures différentes

Procédurale :
C'est une façon d'envisager son code sous la forme d'un enchainement de procédures ou d'étapes qui vont résoudre les problèmes un par un

fonctionnelle :
La programmation fonctionnelle est une façon de concevoir un code en utilisant un enchainement de fonctions « pures », c'est-à-dire des fonctions qui vont toujours retourner le même résultat si on leur passe les mêmes arguments

Les prototypes / L'Orienté Objet en JavaScript (2/12)

Paradigmes de programmation

Orientée objet:
a programmation orientée objet est une façon de concevoir un code autour du concept d'objets. Un objet est une entité qui peut être vue comme indépendante et qui va contenir un ensemble de variables (qu'on va appeler propriétés) et de fonctions (qu'on appellera méthodes). Ces objets vont pouvoir interagir entre eux,

le JavaScript, en particulier, supporte chacun des trois paradigmes principaux cités ci-dessus ce qui signifie qu'on va pouvoir coder en procédural, en fonctionnel et en orienté objet en JavaScript ;

Les prototypes / L'Orienté Objet en JavaScript (3/12)

Création d'un objet JavaScript littéral

- Nous pouvons créer des objets de 4 manières différentes en JavaScript:
  - Créer un objet littéral
  - Utiliser le constructeur Object()
  - Utiliser une fonction constructeur personnalisée
  - Utiliser la méthode Create()

Exemple de création d'objet littéral:

```
/*"pierre" est une variable qui contient un objet. Par abus de langage,
 *on dira que notre variable EST un objet*/
let pierre = {
    //nom, age et mail sont des propriétés de l'objet "pierre"
    nom : ['Pierre', 'Giraud'],
    age : 29,
    mail : 'pierre.giraud@edhec.com',

    //Bonjour est une méthode de l'objet pierre
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
};
```

Les prototypes / L'Orienté Objet en JavaScript (4/12)

Création d'un objet JavaScript littéral

- Exemple de création d'objet littéral:

```
/*"pierre" est une variable qui contient un objet. Par abus de langage,
 *on dira que notre variable EST un objet*/
let pierre = {
    //nom, age et mail sont des propriétés de l'objet "pierre"
    nom : ['Pierre', 'Giraud'],
    age : 29,
    mail : 'pierre.giraud@edhec.com',

    //Bonjour est une méthode de l'objet pierre
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
};
```

- Pour accéder aux membres d'un objet, les modifier ou en définir de nouveaux,
  - Utiliser le point : pierre.nom, pierre.age
  - Utiliser les crochets : pierre['nom'] , pierre['age']

- L'utilisation du mot clef this : Le mot clef this est un mot clef qui apparait fréquemment dans les langages orientés objets. Dans le cas présent, il sert à faire référence à l'objet qui est couramment manipulé.

Les prototypes / L'Orienté Objet en JavaScript (5/12)

Constructeur d'objets en JavaScript

❑ fonction constructeur d'objets

- Une fonction constructeur d'objets est une fonction qui va nous permettre de créer des objets semblables. En JavaScript, n'importe quelle fonction va pouvoir faire office de constructeur d'objets.
- Pour construire des objets à partir d'une fonction constructeur, nous allons devoir suivre deux étapes : il va déjà falloir définir notre fonction constructeur et ensuite nous allons appeler ce constructeur avec une syntaxe un peu spéciale utilisant le mot clefs new

❑ Exemple:

```
//Utilisateur() est une fonction constructeur
function Utilisateur(n, a, m){
    this.nom = n;
    this.age = a;
    this.mail = m;

    this.bonjour = function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
}

let pierre = new Utilisateur(['Pierre', 'Giraud'], 29, 'pierre.giraud@edhec.com');
```

Les prototypes / L'Orienté Objet en JavaScript (6/12)

Le prototype en JavaScript orienté objet

- Le JavaScript est un langage orienté objet basé sur la notion de prototypes.
- il existe deux grands types de langages orientés objet : ceux basés sur les classes, et ceux basés sur les prototypes
- Dans les langages orientés objet basés sur les classes, tous les objets sont créés à partir de classes et vont hériter des propriétés et des méthodes définies dans la classe.
- Dans les langages orientés objet utilisant des prototypes comme le JavaScript, tout est objet et il n'existe pas de classes et l'héritage va se faire au moyen de prototypes.
- les fonctions en JavaScript sont avant tout des objets. Lorsqu'on créé une fonction, le JavaScript va automatiquement lui ajouter une propriété prototype qui ne va être utile que lorsque la fonction est utilisée comme constructeur, c'est-à-dire lorsqu'on l'utilise avec la syntaxe new
- Cette propriété prototype possède une valeur qui est elle-même un objet. On parlera donc de « prototype objet » ou « d'objet prototype » pour parler de la propriété prototype
- le contenu de la propriété prototype d'un constructeur va être partagé par tous les objets créés à partir de ce constructeur. Comme cette propriété est un objet, on va pouvoir lui ajouter des propriétés et des méthodes que tous les objets créés à partir du constructeur vont partager. Cela permet l'héritage en orienté objet JavaScript

Les prototypes / L'Orienté Objet en JavaScript (7/12)

Le prototype en JavaScript orienté objet

❑ Exemple :

```
//Utilisateur() est une fonction constructeur
function Utilisateur(n, a, m){
    this.nom = n;
    this.age = a;
    this.mail = m;
}

/*On ajoute des propriétés et méthodes au prototype de Utilisateur de la même
 *façon que pour n'importe quel objet*/
Utilisateur.prototype.taille = 170;
Utilisateur.prototype.bonjour = function(){
    alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
};

//Crée deux objets pierre et mathilde en utilisant le constructeur
let pierre = new Utilisateur(['Pierre', 'Giraud'], 29, 'pierre.giraud@edhec.com');
let mathilde = new Utilisateur(['Math', 'Ml'], 27, 'math@edhec.com');
```

Ici, on ajoute une propriété taille et une méthode bonjour() à la propriété prototype du constructeur Utilisateur(). Chaque objet créé à partir de ce constructeur va avoir accès à cette propriété et à cette méthode.

Les prototypes / L'Orienté Objet en JavaScript (8/12)

Le prototype en JavaScript orienté objet

- Par défaut, la propriété prototype d'un constructeur ne contient que deux propriétés :
  - une propriété constructor qui renvoie vers le constructeur contenant le prototype,
  - une propriété __proto__ qui contient elle-même de nombreuses propriétés et méthodes.
- Lorsqu'on crée un objet à partir d'un constructeur, le JavaScript va également ajouter automatiquement une propriété __proto__ à l'objet créé.
- La propriété __proto__ de l'objet créé va être égale à la propriété __proto__ du constructeur qui a servi à créer l'objet.

Les prototypes / L'Orienté Objet en JavaScript (9/12)

La chaine des prototypes et l'objet Object

- Comment un objet peut-il accéder à une propriété ou à une méthode définie dans un autre objet ?
  - lorsqu'on essaie d'accéder à un membre d'un objet, le navigateur (qui exécute le JavaScript) va d'abord chercher ce membre au sein de l'objet
  - S'il n'est pas trouvé, alors le membre va être cherché au sein de la propriété prototype du constructeur qui a servi à créer l'objet.
  - Si le membre a été défini dans la propriété prototype du constructeur alors il est utilisé,
  - Si ce n'est pas le cas alors on va aller chercher dans le prototype du constructeur du constructeur.
  - On dit alors qu'on « remonte la chaine des prototypes »

- Tous les objets en JavaScript descendent par défaut d'un objet de base qui s'appelle Object
  - Cet objet est l'un des objets JavaScript prédéfinis et permet notamment de créer des objets génériques vides grâce à la syntaxe new Object()
  - L'objet ou le constructeur Object va être le parent de tout objet en JavaScript (sauf certains objets particuliers créés intentionnellement pour ne pas dépendre d' Object et également posséder une propriété prototype

Les prototypes / L'Orienté Objet en JavaScript (10/12)

La chaine des prototypes et l'objet Object

Les prototypes / L'Orienté Objet en JavaScript (11/12)

Mise en place d'une hiérarchie d'objets avec héritage en JavaScript

- Pour mettre en place un héritage ou plus exactement un système de délégation (qui est un mot beaucoup plus juste que le terme « héritage » dans le cas du JavaScript), nous allons toujours procéder en trois étapes :
  1. On va déjà créer un constructeur qui sera notre constructeur parent
  2. On va ensuite un constructeur enfant qui va appeler le parent ;
  3. On va modifier la __proto__ de la propriété prototype de l'enfant pour qu'elle soit égale au parent.

Les prototypes / L'Orienté Objet en JavaScript (12/12)

Mise en place d'une hiérarchie d'objets avec héritage en JavaScript

- Exemple :

```
function Ligne(longueur){
    this.longueur = longueur;
}
Ligne.prototype.taille = function(){
    document.getElementById('p1').innerHTML = 'Longueur : ' + this.longueur};

function Rectangle(longueur, largeur){
    Ligne.call(this, longueur);
    this.largeur = largeur;
}
Rectangle.prototype = Object.create(Ligne.prototype);
Rectangle.prototype.constructor = Rectangle;
Rectangle.prototype.aire = function(){
    document.getElementById('p2').innerHTML =
    'Aire : ' + this.longueur * this.largeur};

function Parallelepipede(longueur, largeur, hauteur){
    Rectangle.call(this, longueur, largeur);
    this.hauteur = hauteur;
}
Parallelepipede.prototype = Object.create(Rectangle.prototype);
Parallelepipede.prototype.constructor = Parallelepipede;
Parallelepipede.prototype.volume = function(){
    document.getElementById('p3').innerHTML =
    'Volume : ' + this.longueur * this.largeur * this.hauteur};

let geo = new Parallelepipede(5, 4, 3);
geo.volume();
geo.aire();
geo.taille();
```

Valeurs primitives et objets globaux JavaScript (1/5)

les types de valeurs

- En JavaScript, il existe 7 types de valeurs différents :
  - string ou « chaine de caractères » en français ;
  - number ou « nombre » en français ;
  - boolean ou « booléen » en français ;
  - null ou « nul / vide » en français;
  - undefined ou « indéfini » en français ;
  - symbol ou « symbole » en français ;
  - object ou « objet » en français ;

Les valeurs appartenant aux 6 premiers types de valeurs sont appelées des valeurs primitives. Les valeurs appartenant au type object sont des objets.

Valeurs primitives et objets globaux JavaScript (2/5)

Définition des valeurs primitives et différence avec les objets

- Le JavaScript possède deux grandes catégories de types de données : les valeurs primitives et les objets.
- On appelle valeur primitive en JavaScript une valeur qui n'est pas un objet et qui ne peut pas être modifiée.
- Concernant les objets : on va pouvoir modifier les membres d'un objet.

- Une différence notable entre valeurs primitives et objets : les valeurs primitives sont passées et comparées par valeur tandis que les objets sont passés et comparés par référence.
  - Si deux valeurs primitives ont la même valeur, elles vont être considérées égales.
  - Pour que deux objets soient égaux, il faut que les deux fassent référence aux mêmes membres

Valeurs primitives et objets globaux JavaScript (3/5)

L'objet global JavaScript String

- Les propriétés de l'objet String:
  - une propriété : length qui permet d'obtenir la longueur d'une chaine de caractères
  - une propriété prototype
- Les méthodes de l'objet String (voir tableau)

Valeurs primitives et objets globaux JavaScript (4/5)

L'objet global JavaScript Number

- Les propriétés de l'objet Number
:
  - La plupart des propriétés de l'objet Number sont des propriétés dites statiques. Cela signifie qu'on ne va pouvoir les utiliser qu'avec l'objet Number en soi et non pas avec une instance de Number()
- Les propriétés à connaitre sont les suivantes :
  - MIN_VALUE et MAX_VALUE représentent respectivement les plus petite valeur numérique positive et plus grand valeur numérique qu'il est possible de représenter en JavaScript ;
  - MIN_SAFE_INTEGER et MAX_SAFE_INTEGER représentent respectivement le plus petit et le plus grand entiers représentables correctement ou de façon « sûre » en JavaScript. L'aspect « sûr » ici faire référence à la capacité du JavaScript à représenter exactement ces entiers et à les comparer entre eux. Au-delà de ces limites, les entiers différents seront jugés égaux ;
  - NEGATIVE_INFINITY et POSITIVE_INFINITY servent respectivement à représenter l'infini côté négatif et côté positif ;
  - NaN représente une valeur qui n'est pas un nombre (« NaN » est l'abréviation de « Not a Number ») et est équivalente à la valeur

Valeurs primitives et objets globaux JavaScript (5/5)

L'objet global JavaScript Number

```
document.getElementById('p1').innerHTML =
    'MIN_VALUE : ' + Number.MIN_VALUE
    + '<br>MAX_VALUE : ' + Number.MAX_VALUE
    + '<br>MIN_SAFE_INTEGER : ' + Number.MIN_SAFE_INTEGER
    + '<br>MAX_SAFE_INTEGER : ' + Number.MAX_SAFE_INTEGER
    + '<br>NEGATIVE_INFINITY : ' + Number.NEGATIVE_INFINITY
    + '<br>POSITIVE_INFINITY : ' + Number.POSITIVE_INFINITY
    + '<br>NaN : ' + Number.NaN;
```

MIN_VALUE : 5e-324
MAX_VALUE : 1.7976931348623157e+308
MIN_SAFE_INTEGER : -9007199254740991
MAX_SAFE_INTEGER : 9007199254740991
NEGATIVE_INFINITY : -Infinity
POSITIVE_INFINITY : Infinity
NaN : NaN

Manipulation du BOM (1/7)

Définition et présentation des API JavaScript

- Une API (Application Programming Interface ou Interface de Programmation Applicative en français) est une interface, c'est-à-dire un ensemble de codes grâce à laquelle un logiciel fournit des services à des clients.
  - Le principe et l'intérêt principal d'une API est de permettre à des personnes externes de pouvoir réaliser des opérations complexes et cachant justement cette complexité.
  - Une API peut être comparée à une commande de voiture (pédale d'accélération, essuie-glace, etc.) : lorsqu'on accélère ou qu'on utilise nos essuies glace, on ne va pas se préoccuper de comment la voiture fait pour effectivement avancer ou comment les essuies glace fonctionnent. On va simplement se contenter d'utiliser les commandes (l'API) qui vont cacher la complexité des opérations derrière et nous permettre de faire fonctionner la voiture (le logiciel tiers).

Manipulation du BOM (2/7)

Définition et présentation des API JavaScript

- Les API JavaScript vont pouvoir être classées dans deux grandes catégories :
  - Les API intégrées aux navigateurs web et qu'on va donc pouvoir utiliser immédiatement pour du développement web comme l'API DOM (Document Object Model) qui va nous permettre de manipuler le HTML et le CSS d'une page, l'API Geolocation qui va nous permettre de définir des données de géolocalisation ou encore l'API Canvas qui permet de dessiner et de manipuler des graphiques dans une page ;
  - Les API externes, proposées par certains logiciels ou sites comme la suite d'API Google Map qui permettent d'intégrer et de manipuler des cartes dans nos pages web ou encore l'API Twitter qui permet d'afficher une liste de tweets sur un site par exemple ou bien l'API YouTube qui permet d'intégrer des vidéos sur un site.

Manipulation du BOM (3/7)

Browser Object Model (BOM) et l'objet Window

- Le BOM est une sorte de « super API » elle-même composée de plusieurs API dont certaines sont elles mêmes composées de plusieurs API et etc.
- A la base du BOM, nous avons l'interface Window qui représente une fenêtre de navigateur contenant une page ou un document.
- L'objet Window est supporté par tous les navigateurs et tous les objets globaux, variables globales et fonctions globales appartiennent automatiquement à cet objet (c'est-à-dire sont des enfants de cet objet).
- Dans un navigateur utilisant des onglets, comme Firefox, chaque onglet contient son propre objet Window
- Cet objet Window est un objet dit « implicite » : nous n'aurons généralement pas besoin de le mentionner de manière explicite pour utiliser les méthodes (ou fonctions globales) et propriétés (ou variables globales) lui appartenant.

Manipulation du BOM (4/7)

Browser Object Model (BOM) et l'objet Window

- Les objets suivants appartiennent au BOM et sont tous des enfants de Window :
  - L'objet Navigator qui représente l'état et l'identité du navigateur et qui est utilisé avec l'API Geolocation
  - L'objet History qui permet de manipuler l'historique de navigation du navigateur
  - L'objet Location qui fournit des informations relatives à l'URL de la page courante ;
  - L'objet Screen qui nous permet d'examiner les propriétés de l'écran qui affiche la fenêtre courante ;
  - L'objet Document et le DOM dans son ensemble que nous étudierons en détail dans la suite.

- Les propriétés de l'objet Window :
  - Les propriétés outerHeight et outerWidth vont retourner la hauteur et la largeur de la fenêtre du navigateur en comptant les options du navigateur
  - Les propriétés innerHeight et innerWidth vont retourner la hauteur et la largeur de la partie visible de la fenêtre de navigation (la partie dans laquelle le code est rendu).

```
document.getElementById('p1').innerHTML =
    'Taille de la fenêtre (ext) : ' + window.outerWidth + '*' + window.outerHeight;

document.getElementById('p2').innerHTML =
    'Taille de la fenêtre (int) : ' + window.innerWidth + '*' + window.innerHeight;
```

Manipulation du BOM (5/7)

Les méthodes de Window

- **Afficher des boites de dialogue dans une fenêtre :**
  - la méthode alert() permet d'afficher une boite d'alerte : window.alert ("Message à afficher "); ou alert("Message à afficher ");
  - la méthode prompt() affiche une boite de dialogue permettant aux utilisateurs de nous envoyer du texte: reponse = window.prompt("texte","chaine par défaut"); ou prompt("texte","chaine par défaut");
  - la méthode confirm() ouvre une boite avec un message (facultatif) et deux boutons pour l'utilisateur : un bouton Ok et un bouton Annuler. Si l'utilisateur clique sur « Ok », le booléen true est renvoyé par la fonction ce qui va donc nous permettre d'effectuer des actions en fonction du choix de l'utilisateur.

Manipulation du BOM (6/7)

Les méthodes de Window

- **Ouvrir, fermer, redimensionner ou déplacer une fenêtre:**
  - la méthode open() nous permet d'ouvrir une certaine ressource dans une fenêtre, un onglet ou au sein d'un élément iframe
  - Syntaxe : window.open(*URL, name, specs*)
    - URL: Si aucune URL n'est spécifiée, une nouvelle fenêtre/onglet vide s'ouvre
    - Name: L'attribut cible ou le nom de la fenêtre. Valeurs possible: _blank, _parent, _self, _top
    - Specs : Optionel, Une liste d'éléments séparés par des virgules, sans espaces.
  - Exemple: Ouvrez une nouvelle fenêtre et contrôlez son apparence :

```
window.open("https://www.w3schools.com", "_blank", "toolbar=yes,scrollbars=yes,resizable=yes,top=500,left=500,width=400,height=400");
```

  - Exemple : Remplacez la fenêtre actuelle par une nouvelle fenêtre :

```
window.open("", "_self");
```

Manipulation du BOM (7/7)

Les méthodes de Window

- **Ouvrir, fermer, redimensionner ou déplacer une fenêtre:**
  - La méthode open() va également renvoyer une référence pointant vers la fenêtre créée qu'on va pouvoir utiliser ensuite avec d'autres méthodes.
    - resizeBy(largeur, hauteur) : redimensionner en ajoutant ou en enlevant à sa taille actuelle un certain nombre de pixels
    - resizeTo(largeur, hauteur) : en lui passant une nouvelle taille
    - moveBy() : déplacer la fenêtre relativement à sa position de départ:
      fenetre.moveBy(100, 100);//Déplace la fenêtre 100px à droite et 100px en bas
    - moveTo() : déplacer de manière absolue, par rapport à l'angle supérieur gauche de l'espace de travail
      fenetre.moveTo(0, 0);//Place la fenêtre contre le bord supérieur gauche
    - scrollBy() :
      fenetre.scrollBy(0, 200);//Défile de 200px vers le bas
    - scrollTo()
      fenetre.scrollTo(0, 0);//Remonte en haut de la page
    - close() : fermer une fenêtre
      fenetre.close();

Manipulation du HTML: le DOM (1/21)

Definition

- Le Modèle Objet de Document, ou DOM, *Document Object Model,* est un outil permettant l'accès aux documents HTML et XML, Il permet deux choses au développeur :
  - Il fournit une représentation structurée du document ;
  - Il codifie la manière dont un script peut accéder à cette structure.

  Il s'agit donc essentiellement d'un moyen de lier une page Web, par exemple, à un langage de programmation ou de script.

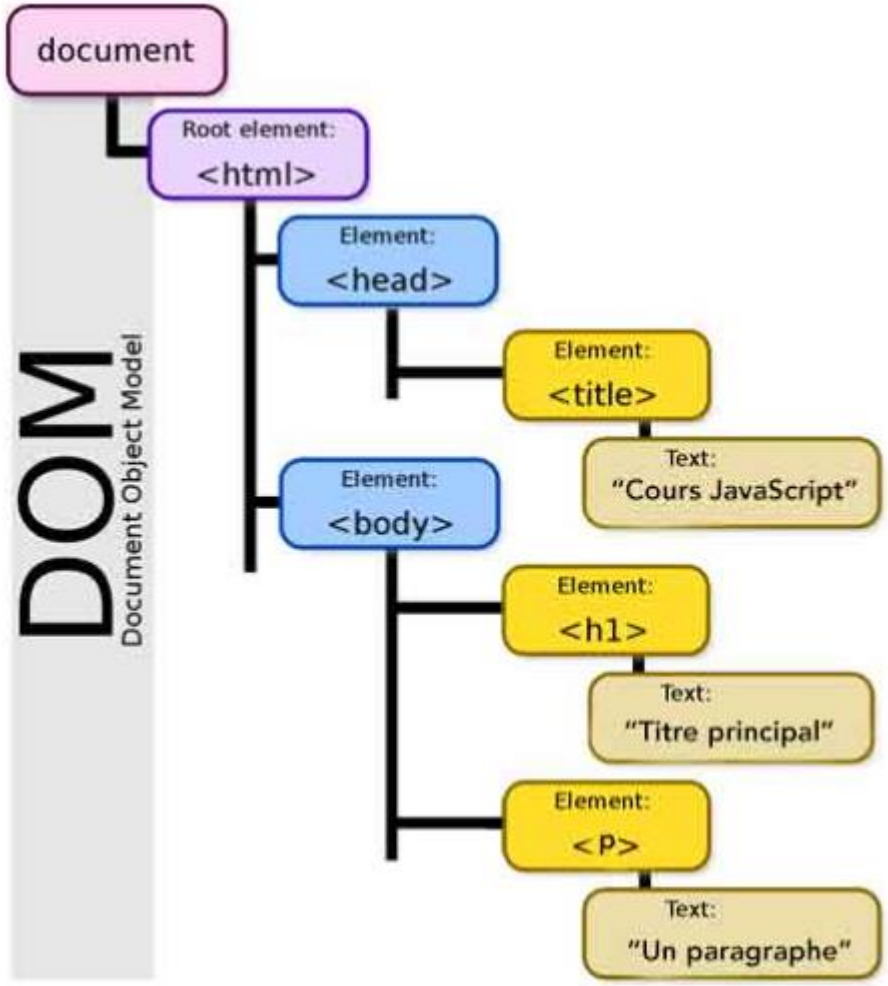
- **Qu'est-ce qu'un nœud**
- Un langage de marquage comme HTML, ou tout autre langage basé sur XML, peut être schématisé comme une arborescence hiérarchisée. Les différentes composantes d'une telle arborescence sont désignés comme étant des nœuds. L'objet central du modèle DOM est pour cette raison l'objet **node** (*node* = nœud) .
- Le terme « nœud » est un terme générique qui sert à désigner tous les objets contenus dans le DOM. A l'extrémité de chaque branche du DOM se trouve un nœud.

Manipulation du HTML: le DOM (2/21)

Exemple de noeud

- I

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours JavaScript</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Titre principal</h1>
        <p>Un paragraphe</p>
    </body>
</html>
```



Manipulation du HTML: le DOM (3/21)

Exemple de noeud

- Exemple des types de nœuds que vous pourrez rencontrer et qui sont représentés par des constantes auxquelles une valeur est liée :

Constante	Valeur	Description
ELEMENT_NODE	1	Représente un nœud élément (comme p ou div par exemple)>
TEXT_NODE	3	Représente un nœud de type texte
PROCESSING_INSTRUCTION_NODE	7	Nœud valable dans le cas d'un document XML. Nous ne nous en préoccuperons pas ici.
COMMENT_NODE	8	Représente un nœud commentaire
DOCUMENT_NODE	9	Représente le nœud formé par le document en soi
DOCUMENT_TYPE_NODE	10	Représente le nœud doctype
DOCUMENT_FRAGMENT_NODE	11	Représente un objet document minimal qui n'a pas de parent (ce type de nœud ne nous intéressera pas ici

Manipulation du HTML: le DOM (4/21)

Accéder aux éléments dans un document avec JavaScript et modifier leur contenu

- Accéder à un élément à partir de son sélecteur CSS associé
- La façon la plus simple d'accéder à un élément dans un document va être de la faire en le ciblant avec le sélecteur CSS qui lui est associé. Deux méthodes nous permettent de faire cela :
  - La méthode querySelector() retourne un objet Element représentant le premier élément dans le document correspondant au sélecteur (ou au groupe de sélecteurs) CSS passé en argument ou la valeur null si aucun élément correspondant n'est trouvé.
  - La méthode querySelectorAll() renvoie un objet appartenant à l'interface NodeList, une liste statique (c'est-à-dire une liste dont le contenu ne sera pas affecté par les changements dans le DOM) des éléments du document qui correspondent au sélecteur (ou au groupe de sélecteurs) CSS spécifiés.
  - Pour itérer dans cette liste d'objets NodeList et accéder à un élément en particulier, on va pouvoir utiliser la méthode forEach(), Cette méthode prend une fonction de rappel en argument et cette fonction de rappel peut prendre jusqu'à trois arguments optionnels qui représentent:
    - L'élément en cours de traitement dans la NodeList
    - L'index de l'élément en cours de traitement dans la
    - L'objet NodeList auquel forEach() est appliqué.

Manipulation du HTML: le DOM (5/21)

Accéder aux éléments dans un document avec JavaScript et modifier leur contenu

Exemple d'utilisation des méthodes querySelector() et querySelectorAll()

```
/*Sélectionne le premier paragraphe du document et change son texte avec la
 *propriété textContent que nous étudierons plus tard dans cette partie*/
document.querySelector('p').textContent = '1er paragraphe du document';

let documentDiv = document.querySelector('div'); //1er div du document
//Sélectionne le premier paragraphe du premier div du document et modifie son texte
documentDiv.querySelector('p').textContent = '1er paragraphe du premier div';

/*Sélectionne le premier paragraphe du document avec un attribut class='bleu'
 *et change sa couleur en bleu avec la propriété style que nous étudierons
 *plus tard dans cette partie*/
document.querySelector('p.bleu').style.color = 'blue';

//Sélectionne tous les paragraphes du document
let documentParas = document.querySelectorAll('p');

//Sélectionne tous les paragraphes du premier div
let divParas = documentDiv.querySelectorAll('p');

/*On utilise forEach() sur notre objet NodeList documentParas pour rajouter du
 *texte dans chaque paragraphe de notre document*/
documentParas.forEach(function(nom, index){
    nom.textContent += ' (paragraphe n°:' + index + ')';
});
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours JavaScript</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
        <script src='cours.js' async></script>
    </head>

    <body>
        <h1 class='bleu'>Titre principal</h1>
        <p id='p1'>Un paragraphe</p>
        <div>
            <p>Un paragraphe dans le div</p>
            <p class='bleu'>Un autre paragraphe dans le div</p>
        </div>
        <p>Un autre paragraphe</p>
    </body>
</html>
```

Manipulation du HTML: le DOM (6/21)

Accéder aux éléments dans un document avec JavaScript et modifier leur contenu

- Accéder à un élément en fonction de la valeur de son attribut id
  - La méthode getElementById() renvoie un objet Element qui représente l'élément dont la valeur de l'attribut id correspond à la valeur spécifiée en argument.

- Accéder à un élément en fonction de la valeur de son attribut class
  - La méthode getElementsByClassName() renvoie une liste des éléments possédant un attribut class avec la valeur spécifiée en argument. La liste renvoyée est un objet de l'interface HTMLCollection qu'on va pouvoir traiter quasiment comme un tableau.

- Accéder à un élément en fonction de son identité
  - La méthode getElementsByTagName() permet de sélectionner des éléments en fonction de leur nom et renvoie un objet HTMLCollection qui consiste en une liste d'éléments correspondant au nom de balise passé en argument.

- Accéder à un élément en fonction de son attribut name
  - La méthode getElementsByName() renvoie un objet NodeList contenant la liste des éléments portant un attribut name avec la valeur spécifiée en argument sous forme d'objet.

Manipulation du HTML: le DOM (7/21)

Accéder au contenu des éléments et le modifier

- Pour récupérer le contenu d'un élément ou le modifier, nous allons pouvoir utiliser l'une des propriétés suivantes :
  - La propriété innerHTML de l'interface Element permet de récupérer ou de redéfinir la syntaxe HTML interne à un élément ;
  - La propriété outerHTML de l'interface Element permet de récupérer ou de redéfinir l'ensemble de la syntaxe HTML interne d'un élément et de l'élément en soi ;
  - La propriété textContent de l'interface Node représente le contenu textuel d'un nœud et de ses descendants. On utilisera cette propriété à partir d'un objet Element
  - La propriété innerText de l'interface Node représente le contenu textuel visible sur le document final d'un nœud et de ses descendants. On utilisera cette propriété à partir d'un objet Element,

Manipulation du HTML: le DOM (8/21)

Accéder au contenu des éléments et le modifier

❑ Exemple :

```
//Accède au contenu HTML interne du div et le modifie
document.querySelector('div').innerHTML +=
    '<ul><li>Elément n°1</li><li>Elément n°2</li></ul>';

//Accède au HTML du 1er paragraphe du document et le modifie
document.querySelector('p').outerHTML = '<h2>Je suis un titre h2</h2>';

/*Accède au contenu textuel de l'élément avec un id='texte' et le modifie.
 *Les balises HTML vont ici être considérées comme du texte*/
document.getElementById('texte').textContent = '<span>Texte modifié</span>';

//Accède au texte visible de l'élément avec l'id = 'p2'
let texteVisible = document.getElementById('p2').innerText;
//Accède au texte (visible ou non) de l'élément avec l'id = 'p2'
let texteEntier = document.getElementById('p2').textContent;

//Affiche les résultats du dessus dans l'élément avec l'id = 'p3'
document.getElementById('p3').innerHTML =
    'Texte visible : ' + texteVisible + '<br>Texte complet : ' + texteEntier;
```

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours JavaScript</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
    </head>

    <body>
        <h1>Titre principal</h1>
        <p id='p1'>Un paragraphe</p>
        <div>
            <p>Un paragraphe dans le div</p>
            <p id='texte'>Un autre paragraphe dans le div</p>
        </div>
        <p id='p2'>Un autre paragraphe
            <span style='visibility: hidden'>avec du contenu caché</span>
        </p>
        <p id='p3'></p>

        <script src='gestioncontenu.js' async></script>
    </body>
</html>
```

Manipulation du HTML: le DOM (9/21)

Naviguer ou se déplacer dans le DOM en JavaScript grâce aux noeuds

- Accéder au parent ou à la liste des enfants d'un nœud :
  - La propriété parentNode de l'interface Node renvoie le parent du nœud spécifié dans l'arborescence du DOM ou null si le nœud ne possède pas de parent.
  - La propriété childNodes de cette même interface renvoie une liste des nœuds enfants de l'élément donné.
  - parentElement : Pour n'accéder au parent que dans le cas où celui-ci est un nœud

- Accéder à un nœud enfant en particulier à partir d'un nœud parent:
  - La propriété firstChild de l'interface Node renvoie le premier nœud enfant direct d'un certain nœud ou null si le nœud ne possède pas de parent.
  - La propriété lastChild au contraire, renvoie le dernier nœud enfant direct d'un certain nœud ou null s'il n'en a pas
  - firstElementChild et lastElementChild : Pour renvoyer le premier et le dernier nœud enfant de type élément seulement d'un certain nœud

Manipulation du HTML: le DOM (10/21)

Naviguer ou se déplacer dans le DOM en JavaScript grâce aux noeuds

- Accéder au nœud précédent ou suivant un nœud dans l'architecture DOM :
  - La propriété previousSibling renvoie le nœud précédent un certain nœud dans l'arborescence du DOM (en ne tenant compte que des nœuds de même niveau) ou null si le nœud en question est le premier.
  - La propriété nextSibling renvoie elle le nœud suivant un certain nœud dans l'arborescence du DOM (en ne tenant compte que des nœuds de même niveau) ou null si le nœud en question est le premier.
  - previousElementSibling et nextElementSibling : Pour accéder spécifiquement au nœud élément précédent ou suivant un certain nœud

- Accéder au nœud précédent ou suivant un nœud dans l'architecture DOM :
  - La propriété nodeName qui retourne une chaine de caractères contenant le nom du nœud (nom de la balise dans le cas d'un nœud de type Element ou #text dans le cas d'un nœud de type Text) ;
  - La propriété nodeValue qui renvoie ou permet de définir la valeur du nœud. On pourra notamment utiliser cette propriété sur des nœuds #text pour obtenir le texte qu'ils contiennent ;
  - La propriété nodeType renvoie un entier qui représente le type du nœud (voir tableau P49).

Manipulation du HTML: le DOM (11/21)

Ajouter, modifier ou supprimer des éléments du DOM avec JavaScript

- Créer de nouveaux nœuds et les ajouter dans l'arborescence du DOM:
  - La méthode createElement() de l'interface Document, Cette méthode va prendre en argument le nom de l'élément HTML que l'on souhaite créer.
  - Pour insérer du texte dans notre nœud élément on va pouvoir par exemple utiliser la propriété textContent

```
let newP = document.createElement('p');
let newTexte = document.createTextNode('Texte écrit en JavaScript');

newP.textContent = 'Paragraphe créé et inséré grâce au JavaScript';
```

- Insérer un nœud dans le DOM:
  - Il existe différentes méthodes qui nous permettent d'insérer des nœuds dans d'autres nœuds. La différence entre ces méthodes va souvent consister dans la position où le nœud va être inséré.
  - les méthodes prepend() et append() vont respectivement nous permettre d'insérer un nœud ou du texte avant le premier enfant d'un certain nœud ou après le dernier enfant de ce nœud.

Manipulation du HTML: le DOM (12/21)

Ajouter, modifier ou supprimer des éléments du DOM avec JavaScript

- les méthodes insertAdjacentElement(), insertAdjacentText() et insertAdjacentHTML() de l'interface Element peuvent être utilisées pour insérer des nœuds dans le DOM.
- Pour chacune de ces trois méthodes, nous allons devoir spécifier la position où on souhaite insérer nos nœuds ainsi que le nœud à insérer en arguments. Pour la position, il faudra fournir l'un des mots clefs suivants :
  - beforebegin
  - afterbegin
  - beforeend
  - afterend

```
let b = document.body;
let p1 = document.getElementById('p1');
let p2 = document.getElementById('p2');
let newP = document.createElement('p');
let htmlContent = '<strong> et du texte important</strong>';

newP.textContent = 'Paragraphe créé et inséré grâce au JavaScript';

//Ajoute un paragraphe après p1
p1.insertAdjacentElement('afterend', newP);

//Ajoute le contenu de htmlContent avant la balise fermante de p1
p1.insertAdjacentHTML('beforeend', htmlContent);

//Ajoute du texte après la balise ouvrante de p2
p2.insertAdjacentText('afterbegin', 'Texte ajouté dans ');
```

Manipulation du HTML: le DOM (13/21)

Ajouter, modifier ou supprimer des éléments du DOM avec JavaScript

- Déplacer un nœud dans le DOM :
- Pour déplacer un nœud dans le DOM, on peut utiliser l'une des méthodes appendChild() ou insertBefore() de Node en leur passant en argument un nœud qui existe déjà et qui est déjà placé dans le DOM.

```
let b = document.body;
let p1 = document.getElementById('p1');
let p4 = b.lastElementChild; //On accède au dernier paragraphe

//On déplace p1 juste avant p4 dans le DOM
b.insertBefore(p1, p4);
```

- Pour cloner un nœud, on peut utiliser la méthode cloneNode() de Node qui renvoie une copie du nœud sur lequel elle a été appelée.
- Pour remplacer un nœud, on utilisera plutôt la méthode replaceChild() de cette même interface qui va remplacer un certain nœud par un autre.

Manipulation du HTML: le DOM (14/21)

Ajouter, modifier ou supprimer des éléments du DOM avec JavaScript

- Supprimer un nœud du DOM
  - la méthode removeChild() de Node va supprimer un nœud enfant passé en argument d'un certain nœud parent de l'arborescence du DOM et retourner le nœud retiré.
  - la méthode remove() qui permet tout simplement de retirer un nœud de l'arborescence,

```
let b = document.body;
let p1 = document.getElementById('p1');
let p2 = document.getElementById('p2');

//Supprime p1 du DOM et renvoie le noeud supprimé
let eltDel = b.removeChild(p1);

//Supprime p2 du DOM
p2.remove()

alert('Noeud supprimé du DOM : ' + eltDel + '\nContenu : ' + eltDel.textContent);
```

Manipulation du HTML: le DOM (15/21)

Manipuler les attributs et les styles des éléments via le DOM en JavaScript

- Tester la présence d'attributs :
  - La méthode hasAttribute() nous permet de tester la présence d'un attribut en particulier pour un élément. Cette méthode prend en argument le nom de l'attribut qu'on recherche et renvoie la valeur booléenne true si l'élément possède bien cet attribut ou false sinon
  - Pour vérifier si un élément possède des attributs ou pas (quels qu'ils soient), on utilisera plutôt la méthode hasAttributes()

- Récupérer la valeur ou le nom d'un attribut ou définir un attribut:
  - la méthode getAttributeNames() permet de récupérer que les noms des attributs d'un élément et renvoie les noms des attributs d'un élément sous forme de tableau
  - la méthode setAttribute() permet d'ajouter un nouvel attribut ou changer la valeur d'un attribut existant pour un élément

Manipulation du HTML: le DOM (16/21)

Manipuler les attributs et les styles des éléments via le DOM en JavaScript

- Supprimer un attribut :
  - La methode removeAttribute() de l'interface Element permet de supprimer un attribut d'un élément,

```
let p1 = document.querySelector('p');
p1.removeAttribute('class');
```

- Modifier les styles d'un élément :
  - La propriété style et de HTMLElement va nous permettre de définir les styles en ligne d'un élément (les styles vont être placés dans la balise ouvrante de l'élément directement).
  - La propriété style retourne un objet à partir duquel on va pouvoir utiliser des propriétés JavaScript représentant les propriétés CSS. Ces propriétés respectent la norme lower camel case : elles doivent être écrites sans espace ni tiret, avec une majuscule au début de chaque mot sauf pour le premier : la propriété CSS background-color, par exemple, s'écrira backgroundColor

```
let p1 = document.querySelector('p');
let p2 = document.getElementById('p2');

p1.style.color = 'crimson'; //Nuance de rouge
p1.style.fontSize = '20px';

p2.style.backgroundColor = 'orange';
```

Manipulation du HTML: le DOM (17/21)

La gestion d'évènements en JavaScript et la méthode addEventListener

- Un gestionnaire d'évènements est toujours divisé en deux parties : une partie qui va servir à écouter le déclenchement de l'évènement, et une partie gestionnaire en soi qui va être le code à exécuter dès que l'évènement se produit.

- Aujourd'hui, en JavaScript, il existe trois grandes façons d'implémenter un gestionnaire d'évènements :
  - On peut utiliser des attributs HTML de type évènement
  - On peut utiliser des propriétés JavaScript liées aux évènements ;
  - On peut utiliser la méthode addEventListener()

Manipulation du HTML: le DOM (18/21)

La gestion d'évènements en JavaScript et la méthode addEventListener

❑ Utilisation des attributs HTML pour gérer un évènement

L'idée va être ici d'insérer un attribut HTML lié à l'évènement qu'on souhaite gérer directement dans la balise ouvrante d'un élément à partir duquel on va pouvoir détecter le déclenchement de cet évènement.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours JavaScript</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
        <script src='cours.js' async></script>
    </head>

    <body>
        <h1>Titre principal</h1>
        <p>Un premier paragraphe</p>
        <button onclick="alert('Bouton cliqué')">Cliquez moi !</button>
        <div onmouseover="this.style.backgroundColor='orange'"
             onmouseout="this.style.backgroundColor='white'">
            <p>Un paragraphe dans un div</p>
            <p>Un autre paragraphe dans le div</p>
        </div>
    </body>
</html>
```

- Ces attributs HTML de « type évènement » possèdent souvent le nom de l'évènement qu'ils doivent écouter et gérer précédé par « on » comme par exemple :
  - L'attribut onclick pour l'évènement « clic sur un élément » ;
  - L'attribut Onmouseover pour l'évènement « passage de la souris sur un élément » ;
  - L'attribut onmouseout pour l'évènement « sortie de la souris d'élément » ;

Manipulation du HTML: le DOM (19/21)

La gestion d'évènements en JavaScript et la méthode addEventListener

❑ Utiliser les propriétés JavaScript pour gérer un évènement

Chaque évènement est représenté en JavaScript par un objet basé sur l'interface Event
Ces gestionnaires d'évènements sont des propriétés qui sont de la forme « on » + nom de l'évènement géré, c'est-à-dire qui ont des noms similaires aux attributs HTML vus précédemment.

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Cours JavaScript</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
        <script src='cours.js' async></script>
    </head>

    <body>
        <h1>Titre principal</h1>
        <p>Un premier paragraphe</p>
        <button>Cliquez moi !</button>
        <div>
            <p>Un paragraphe dans un div</p>
            <p>Un autre paragraphe dans le div</p>
        </div>
    </body>
</html>
```

```javascript
//On sélectionne le premier button et le premier div du document
let b1 = document.querySelector('button');
let d1 = document.querySelector('div');

//On utilise les propriétés gestionnaires d'évènement avec nos éléments
b1.onclick = function(){alert('Bouton cliqué')};
d1.onmouseover = function(){this.style.backgroundColor ='orange'};
d1.onmouseout = function(){this.style.backgroundColor='white'};
```

Manipulation du HTML: le DOM (20/21)

La gestion d'évènements en JavaScript et la méthode addEventListener

- Utiliser la méthode addEventListener() pour gérer un évènement :
  - On va passer deux arguments à cette méthode : le nom d'un évènement qu'on souhaite prendre en charge ainsi que le code à exécuter (qui prendra souvent la forme d'une fonction) en cas de déclenchement de cet évènement.
  - Il est possible d'utiliser la méthode addEventListener() pour réagir plusieurs fois et de façon différente à un même évènement ou pour réagir à différents évènements à partir de différents ou d'un même objet Element,

```
//On sélectionne le premier button et le premier div du document
let b1 = document.querySelector('button');
let d1 = document.querySelector('div');

//On utilise la méthode addEventListener pour gérer des évènements
b1.addEventListener('click', function(){alert('Bouton cliqué')});
d1.addEventListener('mouseover', function(){this.style.backgroundColor ='orange'});
d1.addEventListener('mouseover', function(){this.style.fontWeight ='bold'});
d1.addEventListener('mouseout', function(){this.style.backgroundColor='white'});
```

Manipulation du HTML: le DOM (21/21)

La gestion d'évènements en JavaScript et la méthode addEventListener

- Supprimer un gestionnaire d'évènements avec removeEventListener() :
  - La méthode removeEventListener() de l'interface EventTarget va nous permettre de supprimer un gestionnaire d'évènement déclaré avec addEventListener()
  - Pour cela, il va suffire de passer en argument le type d'évènement ainsi que le nom de la fonction passée en argument de addEventListener()

```
//On sélectionne le premier button et le premier div du document
let b1 = document.querySelector('button');
let d1 = document.querySelector('div');

function changeCouleur(){
    this.style.backgroundColor ='orange';
}

//On utilise la méthode addEventListener pour gérer des évènements
b1.addEventListener('click', function(){alert('Bouton cliqué')});
d1.addEventListener('mouseover', changeCouleur);
d1.addEventListener('mouseover', function(){this.style.fontWeight ='bold'});

//On supprime un évènement
d1.removeEventListener('mouseover', changeCouleur);
```

Interactivité (1/2)

Événements reconnus par Javascript

- click : un clic du bouton gauche de la souris sur une cible
- focusin : une activation d'une cible
- focusout : perte du focus d'une cible
- change : une modification du contenue d'une cible
- submit : une soumission d'un formulaire
- load : à la n du chargement d'un élément
- keydown : appui d'une touche clavier
- keyup : relâchement d'une touche clavier
- mousedown : clic souris
- mouseup : relâchement d'un clic souris

Interactivité (2/2)

Événements liés à Window

❑ L'objet Window possède plusieurs méthodes spécifiques pour la gestion d'un compte à rebours :

- setTimeout(instruction , temps) permet de spécifier un compteur de millisecondes associé à une instruction. Après l'intervalle de temps spécifié, une interruption est produite et l'instruction est évaluée.

- setInterval (instruction , temps) permet de spécier un compteur de millisecondes associé à une instruction. L'instruction est évaluée à intervalles réguliers.

- clearTimeout() et clearInterval () annulent un compte à rebours.

```
setTimeout("window.alert(' Hello !') ;", 1000);
```

Conseils de programmation

- RTFM : http://www.ecma-international.org/publications/standards/Ecma-262.htm
- Programmer TRÈS proprement
- Penser Flux et DOM : tous les éléments sont-ils chargés ? Ai-je modifié des nœuds ? Quels événements pour quels actions ?
- Utiliser les Plugins liés aux développement Web
  - Firebugs
  - Web Developer