ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Développement Web:

JavaScript

Professeur:

Nouhaila MOUSSAMMI

n.moussammi@emsi.ma

2024/2025

Gestion des erreurs

Gestion des erreurs

Introduction

-> Une « erreur » se traduit généralement par un comportement inattendu et non voulu du script. Les erreurs peuvent avoir différentes origines :

- Elles peuvent provenir du développeur dans le cas d'erreur de syntaxe dans le code même ;
- Elles peuvent provenir du serveur dans le cas où celui-ci est dans l'incapacité de fournir les ressources (fichiers) demandés ;
- Elles peuvent provenir du navigateur ;
- Elles peuvent encore être créées par un utilisateur qui envoie une valeur non valide par exemple.

```
D: > cours-javascript > JS cours.js
1    prenom;
2    alert('Ce message ne s\'affichera pas');
3
```

Gestion des erreurs

Introduction

L’objet Error :

L’objet error de JavaScript fournit des informations sur l'erreur quand elle se produit. Cet objet fournit deux propriétés name et message.

Exemples d’erreur :

- **RangeError** Nombre hors limite
- **ReferenceError** Référence inconnue
- **SyntaxError** Erreur de syntaxe
- **TypeError** Une erreur de type s'est produite

…

D: > cours-javascript > JS cours.js > ...

```
1   letx = 5;
2   try {
3     x = y + 1;
4     // y n'est pas référencé en mémoire
5   } catch (err) {
6     console.log(err.name); // Output : ReferenceError
7   }
```

Gestion des erreurs

Gérer une erreur avec les blocs try…catch

-> La gestion des exceptions est une partie essentielle de tout programme. En JavaScript, les erreurs sont gérées principalement avec les constructions **try**, **catch**, **finally** et **throw**.

-> lancer une exception avec l'instruction **throw**.

```
1
2   throw new Error("Une erreur s'est produite");
3
```

Une fois qu'une exception est lancée, le flux d'exécution du programme est interrompu et le système commence à chercher un bloc de code pour "attraper" l'exception. Pour cela, le JavaScript dispose d'une syntaxe en deux blocs **try** et **catch** :

- ❖ Le bloc try contient le code susceptible de générer une erreur.
- ❖ Le bloc catch permet d'attraper l'erreur éventuelle et de définir la manière dont elle sera gérée.

Introduction

JS JavaScript

Gérer une erreur avec les blocs try…catch

D: > cours-javascript > JS cours.js > ...

```
1   function div(){
2       let x = prompt('Entrez un premier nombre (numérateur)');
3       let y = prompt('Entrez un deuxième nombre (dénominateur)');
4
5       if(isNaN(x) || isNaN(y) || x == '' || y == ''){
6           throw new Error('Merci de rentrer deux nombres');
7       }else if(y == 0){
8           throw new Error('Division par 0 impossible')
9       }else{
10          alert(x / y);
11      }
12  }
13
14  try{
15      div();
16  }catch(err){
17      alert(err.message);
18  }
19
```

Introduction

Présentation des variables JavaScript : Déclaration

Finally, Bloc : Nettoyage après une exception (finally)

-> Le bloc finally est un bloc optionnel qui doit être placé juste après un try…catch.

-> Ce bloc nous permet de préciser du code qui sera exécuté dans tous les cas, qu'une erreur ou exception ait été générée ou pas. Notez par ailleurs que les instructions dans un bloc finally s'exécuteront même dans le cas où une exception est levée mais n'est pas attrapée à la différence du reste du code.

-> Le bloc finally va être particulièrement utile dans le cas où on veut absolument effectuer certaines opérations même si une exception est levée

Exercice

Exercice : Validation d'un mot de passe

Écris une fonction **validatePassword** qui vérifie si un mot de passe répond aux conditions suivantes :

- Le mot de passe doit avoir au moins 8 caractères.
- Le mot de passe doit contenir au moins un chiffre.

Si l'une de ces conditions n'est pas respectée, la fonction doit lever une exception en utilisant **throw** avec un message d'erreur approprié.

Utilise un bloc **try...catch** pour tester plusieurs mots de passe et afficher un message personnalisé pour chaque cas.

Solution

JS JavaScript

```
function validatePassword(password) {
    if (password.length < 8) {
        throw new Error("Le mot de passe doit contenir au moins 8 caractères.");
    }

    // Vérifier si le mot de passe contient au moins un chiffre
    let containsNumber = false;
    for (let char of password) {
        if (!isNaN(char)) { // Vérifie si le caractère est un chiffre
            containsNumber = true;
            break;
        }
    }

    if (!containsNumber) {
        throw new Error("Le mot de passe doit contenir au moins un chiffre.");
    }

    return "Mot de passe valide.";
}
```

Solution

JS JavaScript

```
// Test de la fonction
const passwords = ["123456", "password", "pass1234", "motdepasse1"];

passwords.forEach((pwd, index) => {
    console.log(`Test ${index + 1} : "${pwd}"`);
    try {
        const result = validatePassword(pwd);
        console.log(result);
    } catch (err) {
        console.error("Erreur : " + err.message);
    } finally {
        console.log("Vérification terminée.\n");
    }
});
```

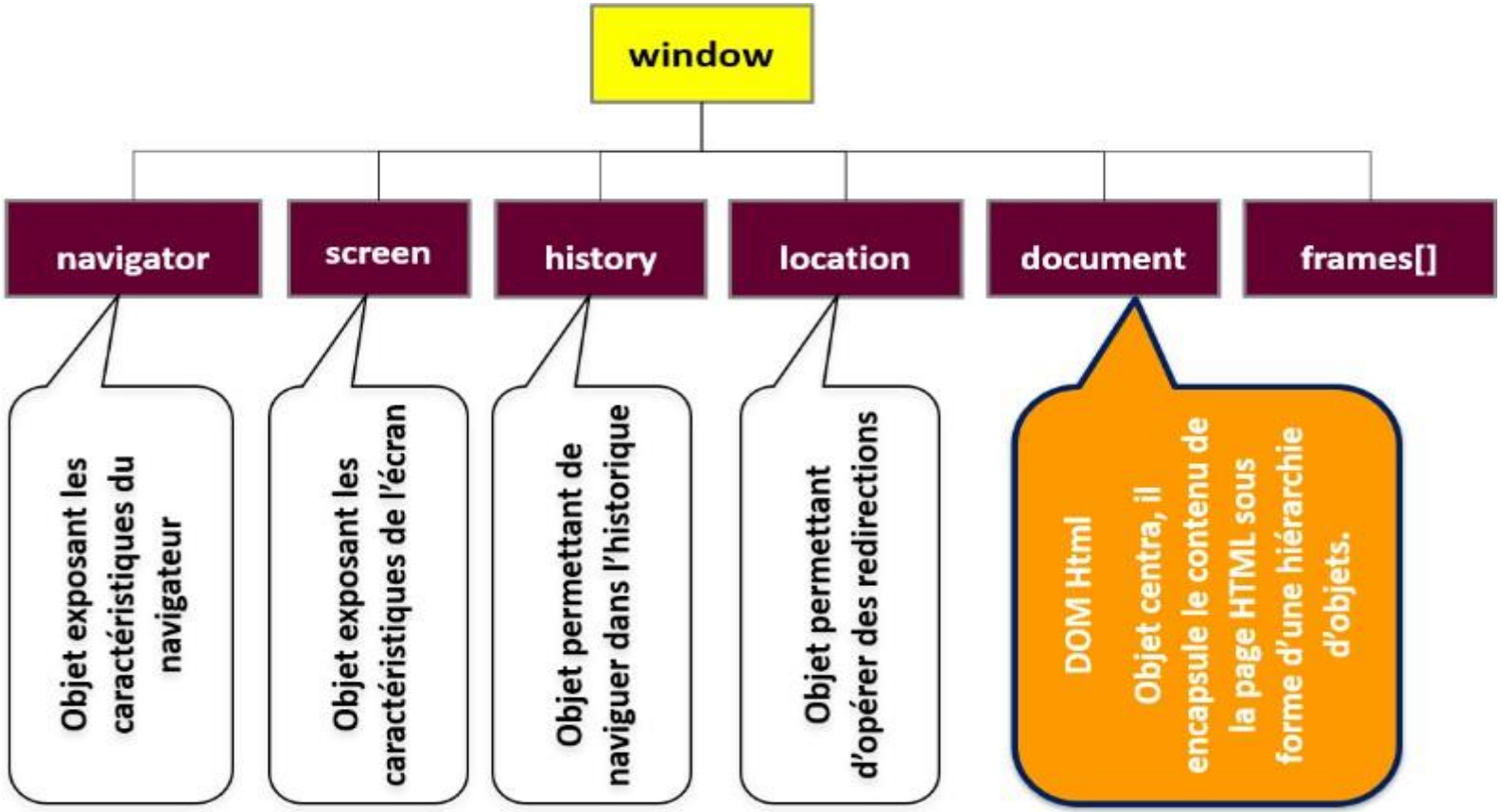
Manipulation du DOM

Manipulation du DOM

JS JavaScript

Introduction au Browser Object Model (BOM)

Le BOM (Browser Object Model) est un ensemble d'objets fournis par le navigateur qui permet aux développeurs d'interagir avec les éléments de la fenêtre du navigateur, comme l'historique, les URL, et les fenêtres, en plus du contenu HTML de la page.



Manipulation du DOM

DOM ou Document Object Model

- Le DOM est une interface de programmation pour des documents HTML qui représente le document (la page web actuelle) sous une forme qui permet aux langages de script comme le JavaScript d'y accéder et d'en manipuler le contenu et les styles.

- Lorsque le navigateur est chargé d'afficher une page Web, il génère automatiquement un modèle objet du document. Ce modèle objet est une représentation en forme d'arborescence de la page, constituée d'objets de type **Node** (nœuds).

- Les navigateurs exploitent cette arborescence pour faciliter la manipulation des éléments, notamment pour appliquer des styles aux bons éléments. De plus, il est possible d'interagir avec ce modèle objet en utilisant un langage de script tel que JavaScript.

Manipulation du DOM

DOM ou Document Object Model

-> Les objets Node ou nœuds du DOM

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.

A partir du nœud racine qui est le nœud HTML on voit que 3 branches se forment : une première qui va aboutir au nœud HEAD, une deuxième qui aboutit à un nœud #text et une troisième qui aboutit à un nœud BODY.

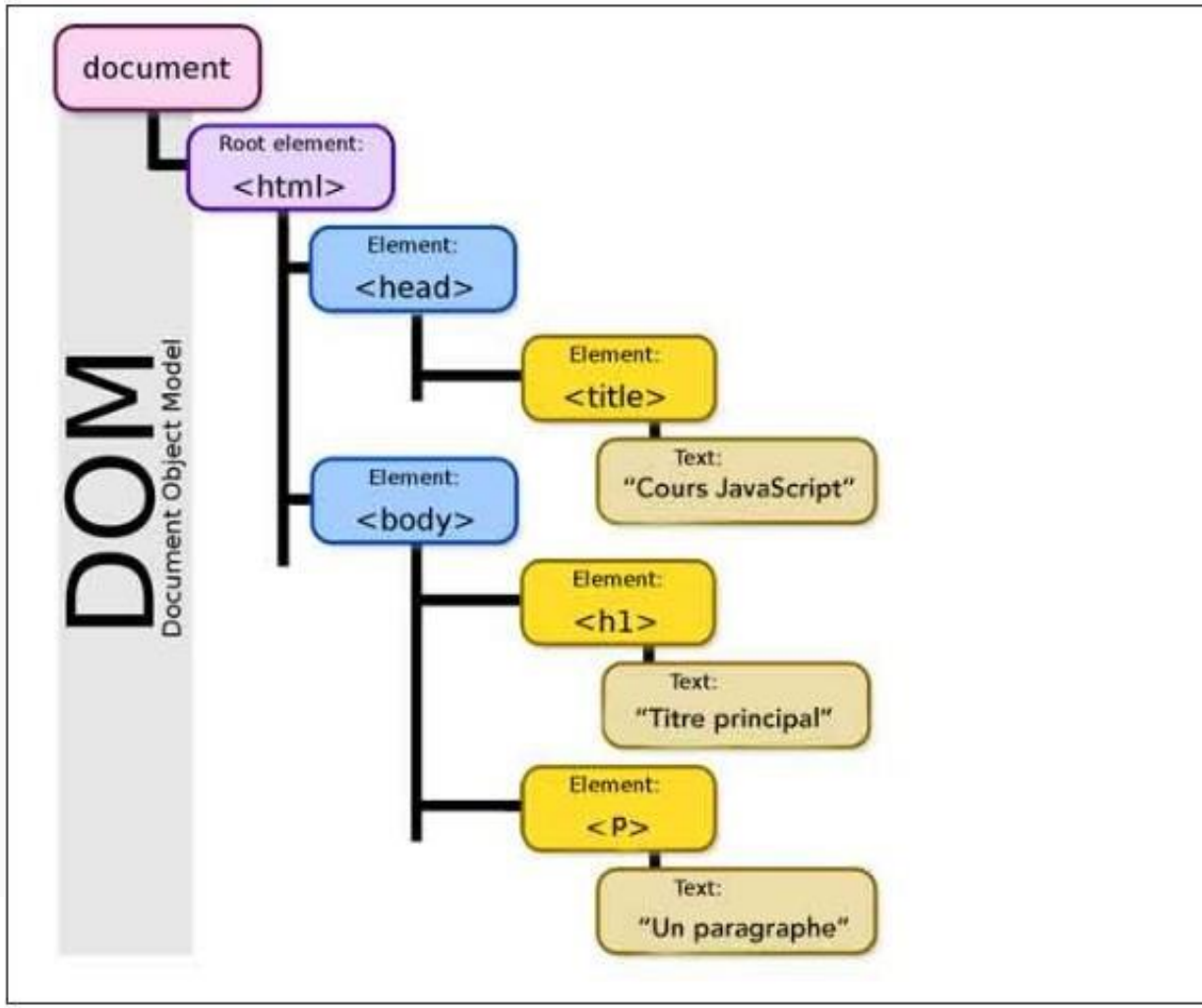
De nouvelles branches se créent ensuite à partir des nœuds HEAD et BODY et etc.

Manipulation du DOM

DOM ou Document Object Model

JS JavaScript

```
<> html.html > ...
1   <!DOCTYPE html>
2   <html>
3       <head>
4           <title>Cours JavaScript</title>
5           <meta charset="utf-8">
6       </head>
7
8       <body>
9           <h1>Titre principal</h1>
10          <p>Un premier paragraphe</p>
11      </body>
12  </html>
13
14
```



Manipulation du DOM

DOM ou Document Object Model

JS JavaScript

-> Pourquoi accéder au DOM ?

Pour faire des opérations usuelles :

- Récupérer la référence d'un élément (et modifier son contenu.)
- Récupérer la référence d'un ensemble d'éléments
- Modifier les attributs d'un élément
- Modifier les styles d'un élément
- Ajouter/supprimer des classes CSS à un élément
- Ajouter un ou plusieurs éléments au DOM
- .....

Manipulation du DOM

DOM ou Document Object Model

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

1. 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.
- A noter que cette liste est mise à jour en direct (ce qui signifie qu'elle sera modifiée dès que l'arborescence DOM le sera).
- Lorsqu'on utilise **getElementsByTagName()** avec un objet Document, la recherche se fait dans tout le document tandis que lorsqu'on utilise **getElementsByTagName()** avec un objet Element, la recherche se fera dans l'élément en question seulement.

Manipulation du DOM

JS JavaScript

DOM ou Document Object Model

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

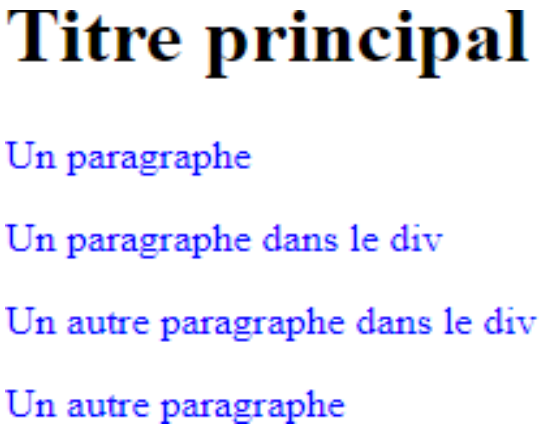
1. Accéder à un élément en fonction de son identité

html.html > ...

```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 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>

```

cours.js > ...

```js
//Sélectionne tous les éléments p du document
let paras = document.getElementsByTagName('p');

//"paras" est un objet de HTMLCollection qu'on va manipuler comme un tableau
for(valeur of paras){
    valeur.style.color = 'blue';
}

```



Manipulation du DOM

DOM ou Document Object Model

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

2. Accéder à un élément en fonction de la valeur de son attribut class

Les interfaces Element et Document vont toutes deux définir une méthode **getElementsByClassName()** qui va renvoyer 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.

Manipulation du DOM

JS JavaScript

DOM ou Document Object Model

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

2. Accéder à un élément en fonction de la valeur de son attribut class

html.html > ...

```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 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>

```

cours.js > ...

```js
//Sélectionne les éléments avec une class = 'bleu'
let bleu = document.getElementsByClassName('bleu');

//"bleu" est un objet de HTMLCollection qu'on va manipuler comme un tableau
for(valeur of bleu){
    valeur.style.color = 'red';
}

```

Titre principal

Un paragraphe

Un paragraphe dans le div

Un autre paragraphe dans le div

Un autre paragraphe

Manipulation du DOM

DOM ou Document Object Model

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

3. Accéder à un élément en fonction de la valeur de son attribut id

getElementById() est une méthode en JavaScript qui permet de sélectionner un élément HTML d'une page web en utilisant son identifiant (id).

C'est un moyen simple d'accéder à un élément en particulier (si celui-ci possède un id) puisque les id sont uniques dans un document.

JS cours.js

```
1   //Sélectionne l'élément avec un id = 'p1' et modifie la couleur du texte
2   document.getElementById('p1').style.color = 'green';
3
```

Titre principal

Un paragraphe

Un paragraphe dans le div

Un autre paragraphe dans le div

Un autre paragraphe

Manipulation du DOM

DOM ou Document Object Model

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

4. Accéder à un élément en fonction de son attribut name

getElementsByName() est une méthode en JavaScript qui permet de récupérer tous les éléments HTML qui ont un attribut name spécifique.

Elle renvoie une collection d'éléments (une **NodeList**) , généralement utilisée dans des formulaires où plusieurs champs partagent le même nom.

Manipulation du DOM

DOM ou Document Object Model

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

4. Accéder à un élément en fonction de son attribut name

```
<form>
  <input type="text" name="username">
  <input type="password" name="password">
  <input type="submit" value="Login">
</form>

<script>
  const inputs = document.getElementsByName("username");
  console.log(inputs); // Retourne une collection d'éléments ayant name="username
  console.log(inputs[0].value); // Accède à la valeur du premier élément
</script>
```

Manipulation du DOM

DOM ou Document Object Model

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

5. 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 : les méthodes **querySelector()** et **querySelectorAll().**

->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**. Les objets **NodeList** sont des collections (des listes) de nœuds.

Manipulation du DOM

JS JavaScript

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

5. Accéder à un élément à partir de son sélecteur CSS associé

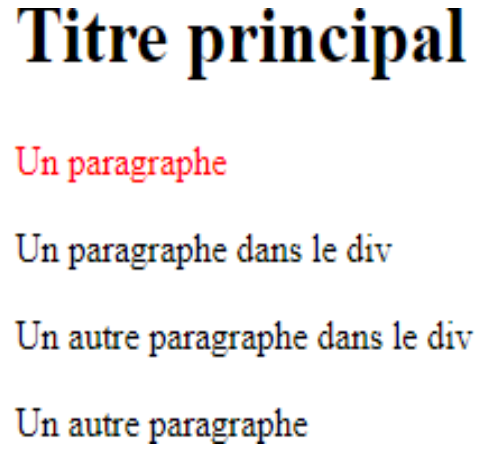
Exemples : (sélection par id)

html.html > ...

```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 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>
```

cours.js > ...

```js
// Sélectionne le paragraphe avec l'id 'p1' et change sa couleur
const paragraphe1 = document.querySelector('#p1');
paragraphe1.style.color = 'red';  // Change la couleur du texte en rouge

```



Manipulation du DOM

DOM ou Document Object Model

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

5. Accéder à un élément à partir de son sélecteur CSS associé

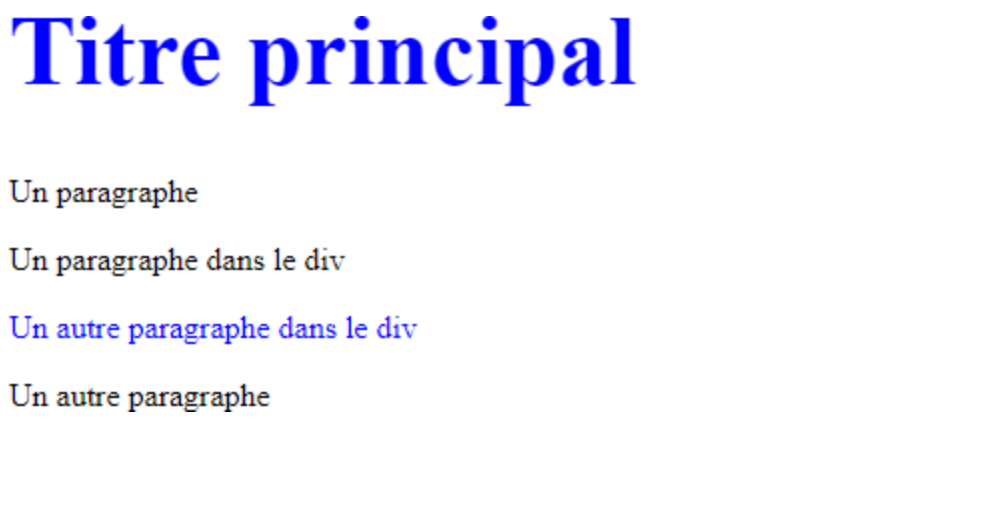
Exemples : (sélection par class) :

html.html > ...

```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 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>

```

cours.js > ...

```js
// Sélectionne le premier élément avec la classe 'bleu' et change sa taille de police
const titreBleu = document.querySelector('.bleu');
titreBleu.style.fontSize = '30px';  // Change la taille de la police à 30px

```



Manipulation du DOM

DOM ou Document Object Model

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

5. Accéder à un élément à partir de son sélecteur CSS associé

Exemples : (sélection par tagname) :

html.html > ...

```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 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>
```

cours.js > ...

```js
// Sélectionne le premier paragraphe et change l'arrière-plan
const premierParagraphe = document.querySelector('p');
premierParagraphe.style.backgroundColor = 'yellow';  // Change l'arrière-plan en jaune

```

Manipulation du DOM

JS JavaScript

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

5. Accéder à un élément à partir de son sélecteur CSS associé

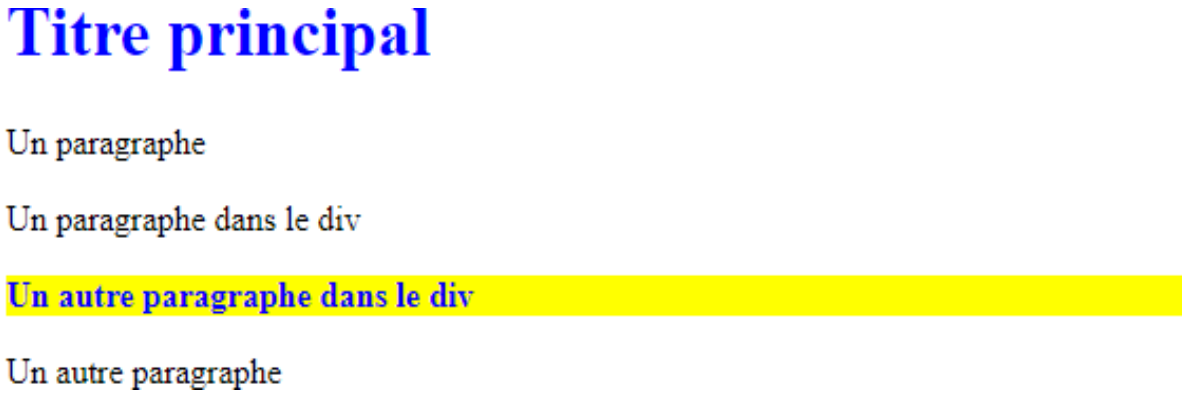
Exemples : (sélection par combinaison de plusieurs critères CSS) :

html.html > ...

```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 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>
```

cours.js > ...

```js
// Sélectionne le paragraphe avec la classe 'bleu' à l'intérieur d'un div
const paragrapheBleuDansDiv = document.querySelector('div p.bleu');
paragrapheBleuDansDiv.style.fontWeight = 'bold';  // Change le texte en gras
paragrapheBleuDansDiv.style.backgroundColor = 'yellow'; //change la couleur de background

```



Manipulation du DOM

DOM ou Document Object Model

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

5. Accéder à un élément à partir de son sélecteur CSS associé

Exemples : (querySelectorAll())

html.html > ...

```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 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>
```

cours.js > ...

```js
// Sélectionne tous les paragraphes <p> de la page
const paragraphes = document.querySelectorAll('p');

// Parcourt chaque paragraphe et change la couleur du texte en vert
paragraphes.forEach(paragraphe => {
  paragraphe.style.color = 'green';
});

```

Manipulation du DOM

DOM ou Document Object Model

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

6. Modifier le contenu d'un nœud :

Jusqu'à présent, nous avons vu différents moyens d'accéder à un élément en particulier dans un document en utilisant le ->DOM.

->Accéder à un nœud en particulier va nous permettre d'effectuer différentes manipulations sur celui-ci, et notamment de récupérer le contenu de cet élément ou de le modifier.

Manipulation du DOM

JS JavaScript

DOM ou Document Object Model

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

6. Modifier le contenu d'un nœud :

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**.

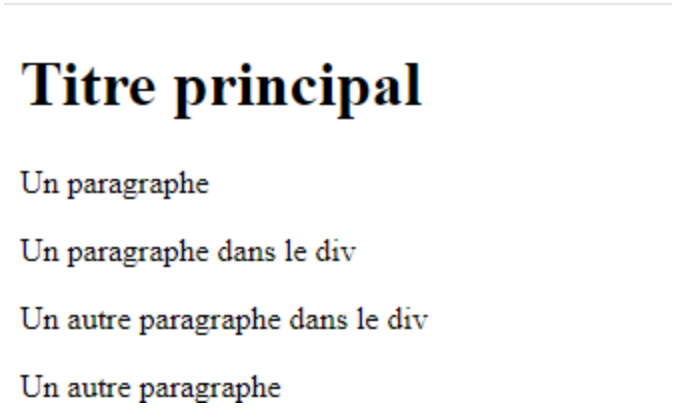
Manipulation du DOM

DOM ou Document Object Model

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

6. Modifier le contenu d'un nœud :

```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 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>
    </body>
</html>
```



Manipulation du DOM

DOM ou Document Object Model

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

6. Modifier le contenu d'un nœud :

cours.js > ...

```
1   //Accède au contenu HTML interne du div et le modifie
2   document.querySelector('div').innerHTML +=
3       '<ul><li>Elément n°1</li><li>Elément n°2</li></ul>';
4
5   //Accède au HTML du 1er paragraphe du document et le modifie
6   document.querySelector('p').outerHTML = '<h2>Je suis un titre h2</h2>';
7
8   /*Accède au contenu textuel de l'élément avec un id='texte' et le modifie.
9    *Les balises HTML vont ici être considérées comme du texte*/
10  document.getElementById('texte').textContent = '<span>Texte modifié</span>';
11
12  //Accède au texte visible de l'élément avec l'id = 'p2'
13  let texteVisible = document.getElementById('p2').innerText;
14  //Accède au texte (visible ou non) de l'élément avec l'id = 'p2'
15  let texteEntier = document.getElementById('p2').textContent;
16
17  //Affiche les résultats du dessus dans l'élément avec l'id = 'p3'
18  document.getElementById('p3').innerHTML =
19      'Texte visible : ' + texteVisible + '<br>Texte complet : ' + texteEntier;
20
```

Titre principal

Je suis un titre h2

Un paragraphe dans le div

Texte modifié

- Elément n°1
- Elément n°2

Un autre paragraphe

Texte visible : Un autre paragraphe
Texte complet : Un autre paragraphe avec du contenu caché