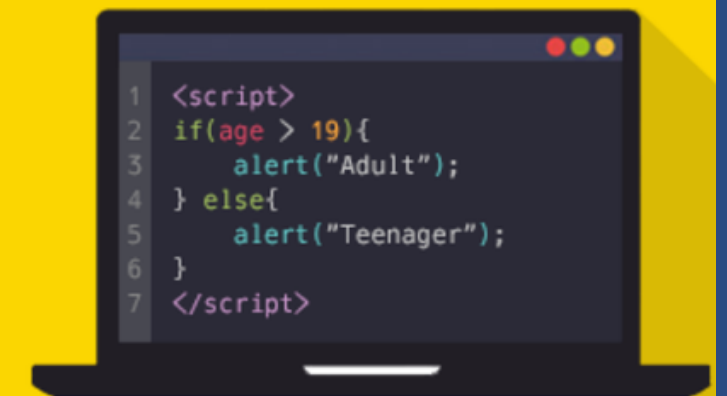


# Javascript

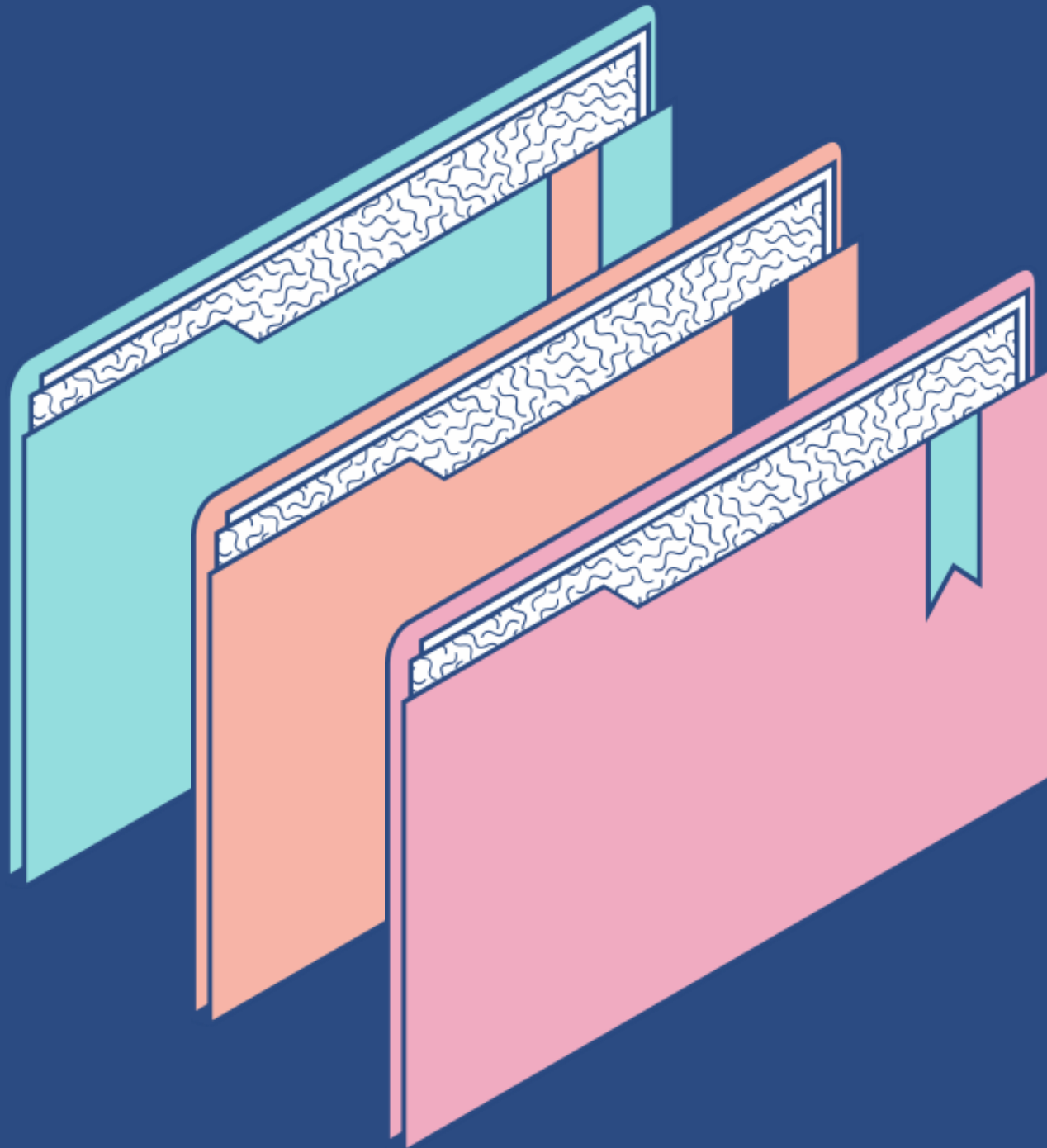


**JavaScript**



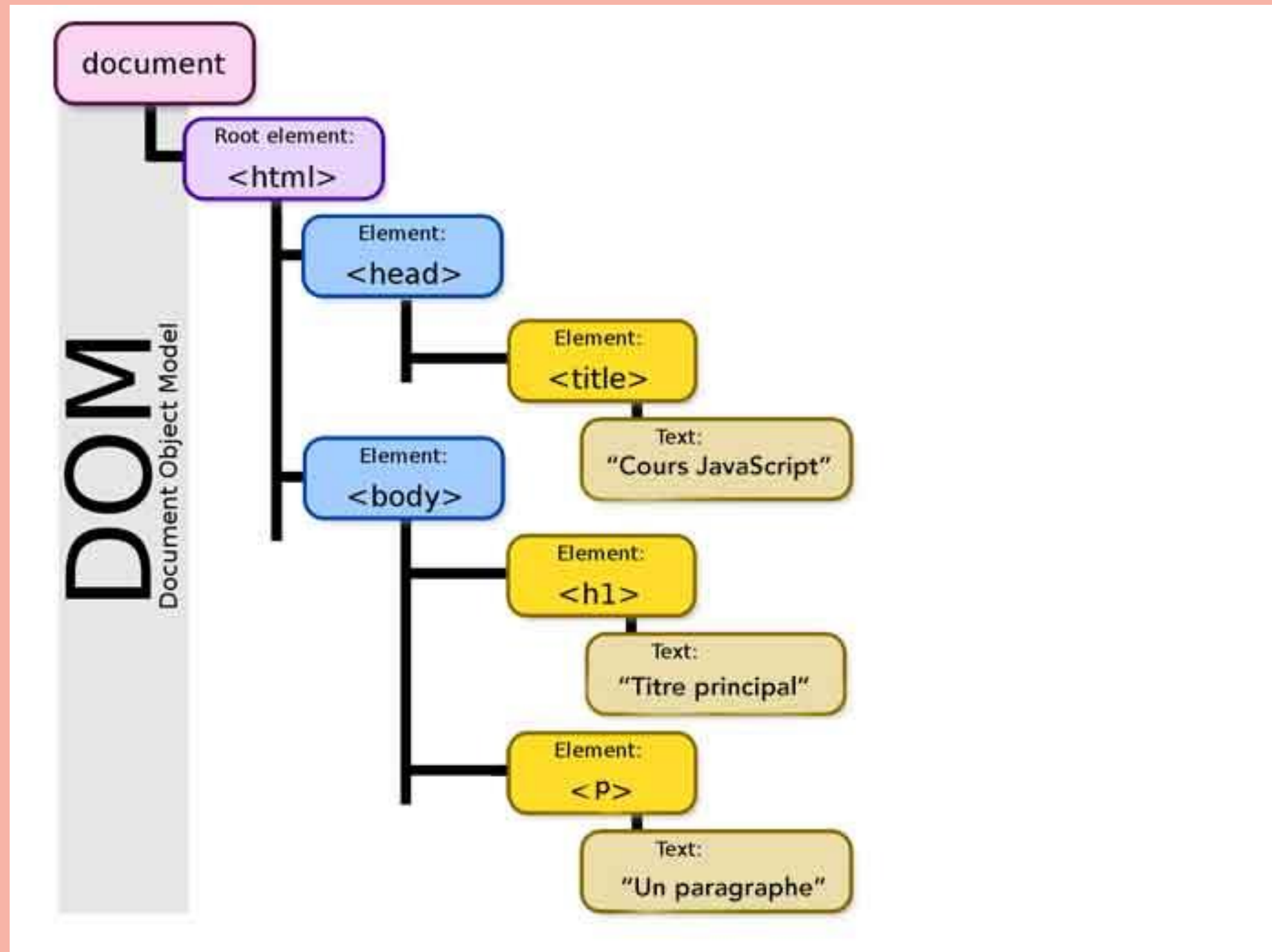
# SOMMAIRE

- Présentation du JS
- Variables
- Structures de contrôle
- Fonctions
- POO en JS
- Valeurs primitives
- Manipulation du BOM
- Manipulation du DOM
- Fonctions avancées
- Stockage de données persistantes
- Canvas
- Asynchrone



# Manipulation du DOM

113





# Présentation DOM

Le DOM est une interface du BOM permettant au JS d'accéder au HTML de la page et de manipuler le contenu et le style des balises.

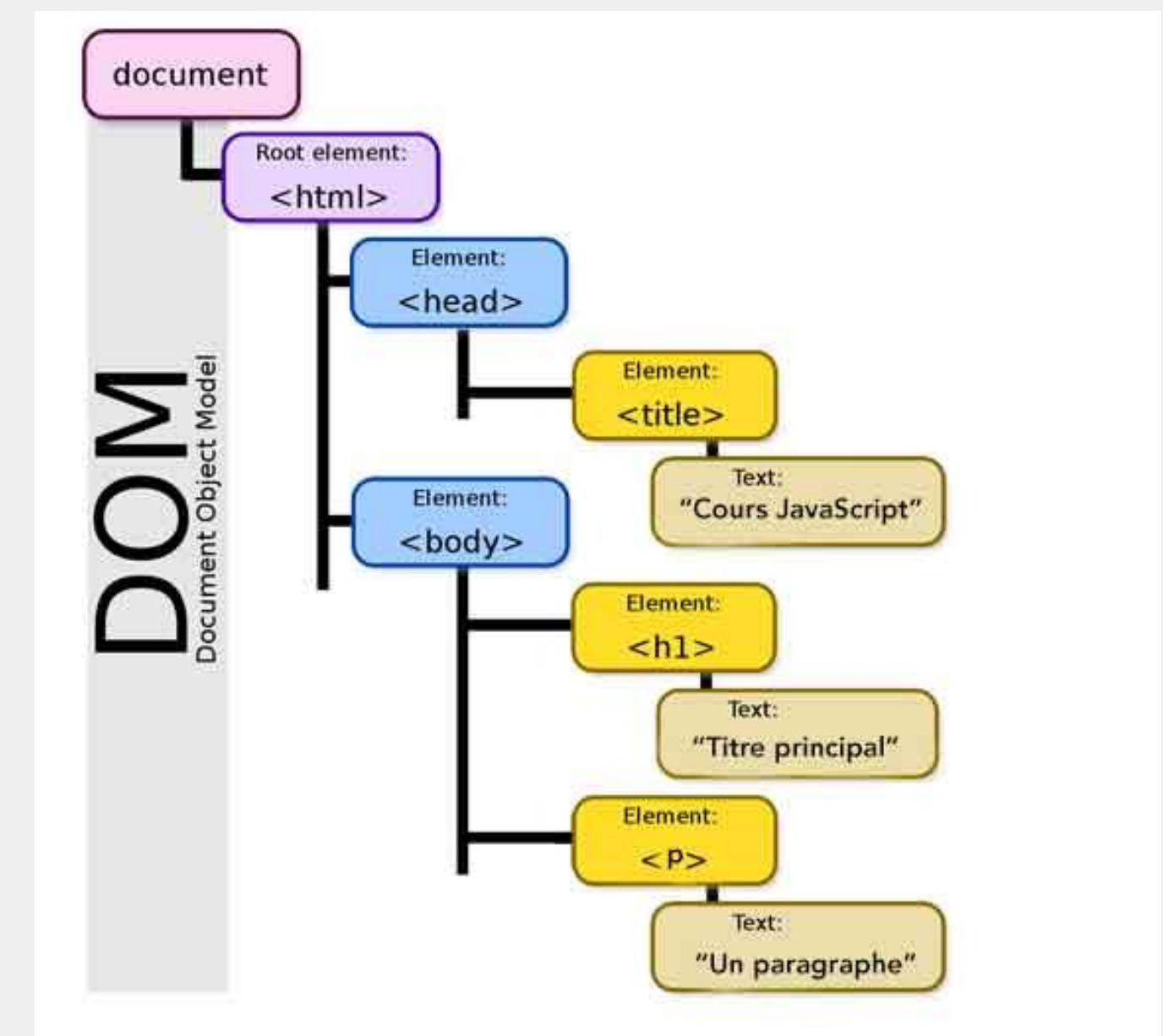
C'est une représentation structurée en arbre de votre page HTML, et créé automatiquement par votre navigateur.

Chaque branche de cet arbre se termine par un "nœud", contenant des objets que nous pourrions utiliser à travers leurs propriétés et méthodes JS.



# Structure du DOM

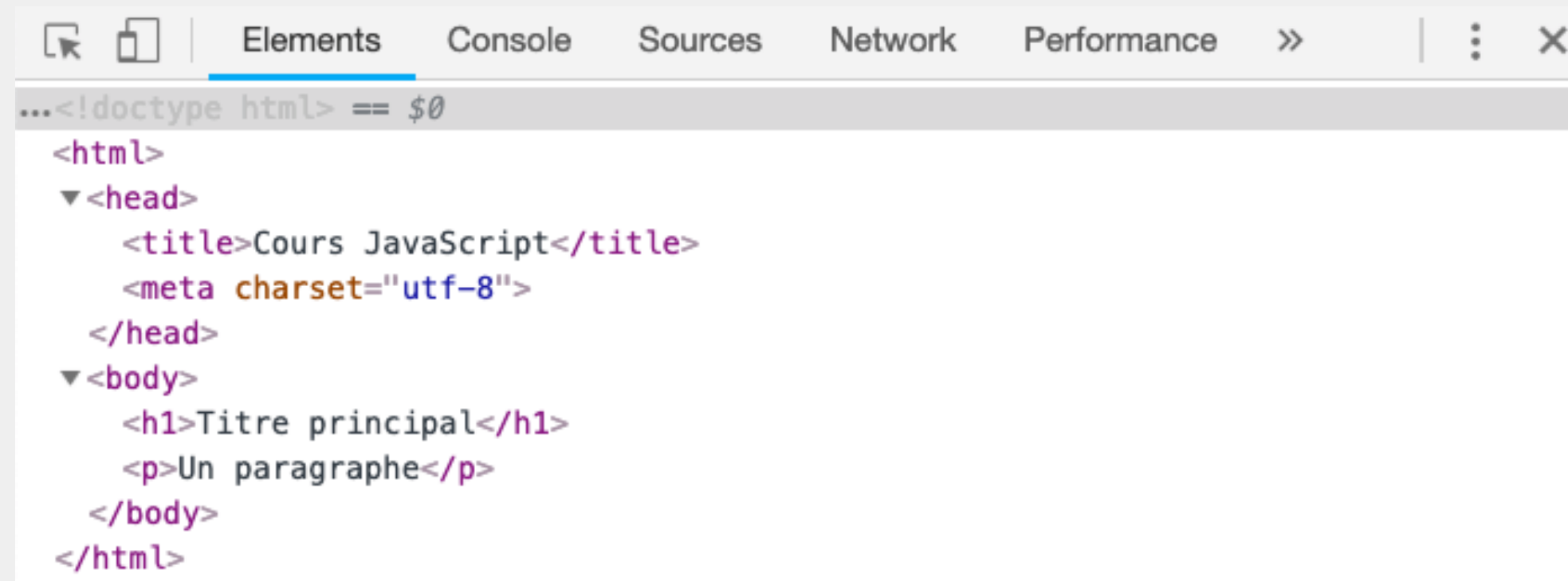
```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Titre principal</h1>
    <p>Un paragraphe</p>
  </body>
</html>
```



# Structure du DOM

La structure DOM reprend la structure créée par les balises HTML, à la différence que des noeuds "texte" sont également mentionnés à la fin de chaque branche.

Le DOM est accessible via l'onglet "Elements" de la console de votre navigateur



```
...<!doctype html> == $0
<html>
  ▼<head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
  </head>
  ▼<body>
    <h1>Titre principal</h1>
    <p>Un paragraphe</p>
  </body>
</html>
```

# Types de noeuds du DOM

On pourra distinguer les nœuds les uns des autres en fonction du type de nœud dont il s'agit, et faire appel à des propriétés et méthodes spécifiques à chacun

Constante	Valeur	Description
ELEMENT_NODE	1	Représente un nœud élément (comme <code>p</code> ou <code>div</code> 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 préoccupons 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)

**Les valeurs non représentées correspondent aux nœuds dépréciés.**





# Interfaces du DOM

Les DOM est composés de plus de 40 interfaces de base, elles mêmes parfois composées d'autres interfaces !

C'est une notion extrêmement complète, mais complexe, dont nous n'étudierons que la surface.





# Interfaces du DOM

Interfaces particulièrement intéressantes :

- L'interface Window qu'on a déjà étudié et qui est liée au DOM
- L'interface Event qui représente tout événement qui a lieu dans le DOM
- L'interface EventTarget qui est une interface que vont implémenter les objets qui peuvent recevoir des événements ;
- L'interface Node qui est l'interface de base pour une grande partie des objets du DOM ;
- L'interface Document qui représente le document actuel et qui va être l'interface la plus utilisée ;
- L'interface Element qui est l'interface de base pour tous les objets d'un document ;

# Accéder à un élément par sélecteur CSS

`querySelector()` : Retourne un objet `Element` représentant le premier élément du document correspondant au sélecteur CSS passé en argument.

`querySelectorAll()` : Retourne un objet `NodeList` renvoyant une liste d'éléments correspondant au sélecteur CSS passé en argument.

Il conviendra ensuite d'itérer cette liste à l'aide de la méthode `forEach()` pour appliquer du contenu à chaque élément.

# Accéder à un élément par sélecteur CSS

```
<!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>
```

```
/*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 + ')';
});
```



# Accéder à un élément par attribut id

getElementById() : permet d'accéder à un élément spécifique identifié par son id.

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

# Accéder à un élément par attribut class

`getElementByClassName()` : renvoi une liste d'objets `HTMLCollection` dont la classe correspondra à celle passée en paramètre.

```
//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 = 'blue';  
}
```

# Accéder à un élément par identité

`getElementByTagName()` : renvoi une liste d'objets `HTMLCollection` dont l'identité correspondra à celle passée en argument.

```
//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';
}
```



# Accéder à des éléments particuliers

L'interface Document fournit des propriétés permettant d'accéder directement à des éléments spécifiques :

- body qui retourne le nœud représentant l'élément body ;
- head qui retourne le nœud représentant l'élément head ;
- links qui retourne une liste de tous les éléments possédant un href avec une valeur ;
  - title qui retourne le titre du document ou permet de le redéfinir ;
  - url qui renvoie l'URL du document sous forme de chaîne de caractères ;
- cookie qui retourne la liste de tous les cookies associés au document sous forme de chaîne de caractères ou qui permet de définir un nouveau cookie.

# Accéder à des éléments particuliers

```
//Sélectionne l'élément body et applique une couleur bleu  
document.body.style.color = 'blue';  
  
//Modifie le texte de l'élément title  
document.title= 'Le Document Object Model';
```



# Accéder et modifier le contenu des éléments

Nous avons vu jusqu'à présent comment récupérer un élément spécifique, ou une liste d'éléments.

Nous allons maintenant nous concentrer sur les façons d'accéder à son contenu et le modifier :

- `innerHTML` : permet de récupérer ou redéfinir le HTML interne d'un élément
- `outerHTML` : permet de récupérer ou redéfinir le HTML interne d'un élément ET de l'élément lui-même.



# Accéder et modifier le contenu des éléments

```
<!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>
```

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

//Accède au HTML du 1er paragraphe du document et le modifie
document.querySelector('p').innerHTML = '<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>';
```

# Exercice d'application

## Accéder aux éléments

En HTML : Créez la page suivante

:

```
<h1>Je suis le titre H1</h1>
<div>
  <p>Je suis le premier paragraphe</p>
  <p id="firstP">Je suis le second paragraphe</p>
  <p class="p">Je suis le troisième paragraphe</p>
</div>
```



# Exercice d'application

## Accéder aux éléments

En JS:

Modifiez vos éléments tels que :

- Le titre H1 ait pour valeur : "Bienvenue sur mon super site !"
- le premier paragraphe devienne un titre H2 affichant "J'adore le développement"
  - Le paragraphe d'id "firstP" affiche : "Le HTML, le CSS ..."
- Le paragraphe de class "p" affiche : affiche : "Et maintenant le JS !"
  - le title affiche : "dev"



# Accéder aux parents et enfants d'un nœud

parentNode : renvoi le parent du nœud spécifié.

childNodes : renvoi une liste des nœuds enfants de l'élément donné.

```
<!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 <span>avec un span</span></p>
    <div>
      <p id='p2'>Un paragraphe dans le div</p>
      <p>Un autre paragraphe dans le div</p>
    </div>
    <p>Un autre paragraphe</p>
  </body>
</html>
```

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

p2.parentNode.style.backgroundColor = 'Rgba(240,160,000,0.5)'; //Orange

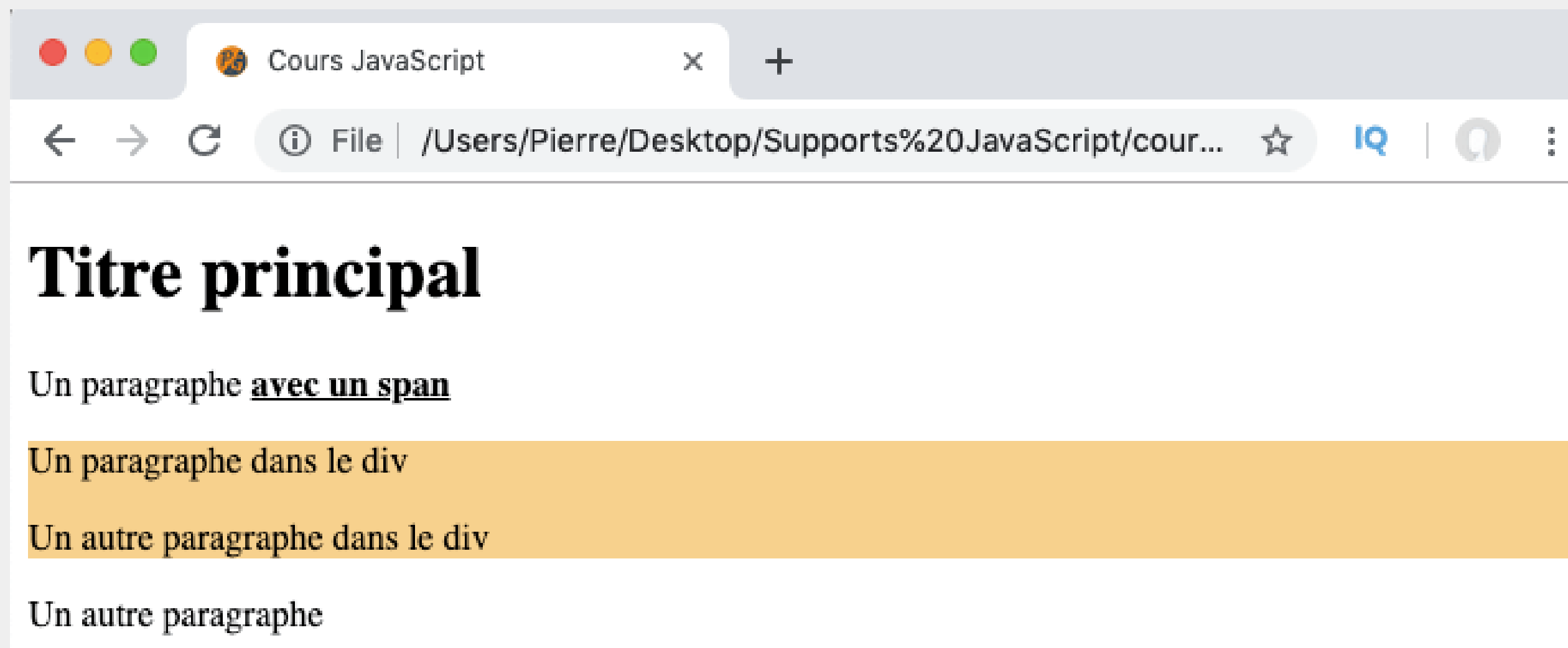
//On accède à tous les noeuds enfants de p1. childNodes renvoie une NodeList
let enfantsP1 = p1.childNodes;

/*On peut ensuite utiliser une boucle forEach() pour tous les manipuler ou
*un indice comme pour les tableaux pour manipuler un noeud enfant en
*particulier (le premier enfant a l'indice 0, le deuxième l'indice 1, etc.)*/
enfantsP1[1].style.fontWeight = 'bold';

/*On accède aux noeuds enfants éléments seulement de p1.
*children renvoie une HTMLCollection*/
let enfantsEltP1 = p1.children;

//On peut ensuite accéder aux différents enfants comme on le ferait avec un tableau
enfantsEltP1[0].style.textDecoration = 'underline';
```

# Accéder aux parents et enfants d'un nœud



# Accéder à un enfant spécifique

firstChild : renvoi le premier noeud enfant d'un noeud donné

lastChild : renvoi le dernier noeud enfant d'un noeud donné

```
//On accède au premier noeud enfant de body
let bodyFirstChild = document.body.firstChild;

//On accède au dernier noeud enfant de body
let bodyLastChild = document.body.lastChild;

//On accède au premier noeud enfant élément de body
let bodyFirstChildElement = document.body.firstChildElement;

//On accède au dernier noeud enfant élément de body
let bodyLastElementChild = document.body.lastElementChild;
```



# Accéder au nœud précédent ou suivant d'un nœud

`previousSibling` : renvoie le noeud précédent un certain de même niveau.

`nextSibling` : renvoie le noeud suivant un certain de même niveau.

```
let p1 = document.getElementById('p1');

let p1PreviousSibling = p1.previousSibling;
let p1NextSibling = p1.nextSibling;
let p1PreviousElementSibling = p1.previousElementSibling;
let p1NextElementSibling = p1.nextElementSibling;

p1PreviousElementSibling.style.color = 'blue';
p1NextElementSibling.style.backgroundColor = 'RGBa(240,160,40,0.5)'; //Orange
```

# Accéder au nom/type/contenu d'un nœud

nodeName : retourne une String contenant le nom du nœud

nodeValue : retourne une String contenant la valeur du nœud

nodeType : retourne un Entier représentant le type de nœud

```
let p1 = document.getElementById('p1');

let p1PreviousSibling = p1.previousSibling;
let p1PreviousElementSibling = p1.previousElementSibling;

alert(
  'Nom noeud p1 : ' + p1.nodeName +
  '\nValeur noeud p1 : ' + p1.nodeValue +
  '\nType noeud p1 : ' + p1.nodeType +
  '\n'
);
```

Nom noeud p1 : P  
Valeur noeud p1 : null  
Type noeud p1 : 1



# Créer un noeud

`createElement()` : Créé un nouvel élément HTML.

```
let newP = document.createElement('p');  
newP.textContent = 'Paragraphe créé et inséré grâce au JavaScript';
```

`createTextNode()` : Créé directement un noeud texte avec du contenu

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



# Insérer un noeud

prepend() : Insère un nœud ou du texte comme premier enfant d'un élément

append() : Insère un nœud ou du texte comme dernier enfant d'un élément

```
let b = document.body;  
let newP = document.createElement('p');  
let newTexte = document.createTextNode('Texte écrit en JavaScript');  
  
newP.textContent = 'Paragraphe créé et inséré grâce au JavaScript';  
  
//Ajoute le paragraphe créé comme premier enfant de l'élément body  
b.prepend(newP);  
  
//Ajoute le texte créé comme dernier enfant de l'élément body  
b.append(newTexte);
```



# Déplacer un noeud

`appendChild()` : prend en argument un noeud qui existe déjà, et le positionne après  
`insertBefore()` : prend en argument un noeud qui existe déjà, et le positionne avant

```
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);
```

# Cloner ou remplacer un noeud

`cloneNode()` : Renvoie une copie du nœud. Passer `true` en argument pour cloner également les enfants, `false` sinon.

`replaceChild()` : Remplace un nœud passé en argument par un autre

```
let b = document.body;
let p1 = document.getElementById('p1');
let p2 = document.getElementById('p2');
let p4 = b.lastElementChild; //On accède au dernier paragraphe
let newP = document.createElement('p'); //On crée un nouveau noeud
newP.textContent = 'Paragraphe créé et inséré grâce au JavaScript';

//On clone p1 et on insère le clone après p2
let cloneP1 = p1.cloneNode(true);
p2.insertAdjacentElement('afterend', cloneP1);

//On remplace p4 par newP
b.replaceChild(newP, p4);
```



# Supprime un nœud

`removeChild()` : supprime un nœud enfant du noeud passé en argument et le Return.  
`remove()` : supprime le noeud.

```
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);
```



# Exercice d'application : Accéder aux nœuds complexes

Créez un nouveau fichier HTML contenant uniquement le doctype.

En JS : Créez le rendu suivant, où tous les éléments à l'exception du titre H1 sont à l'intérieur d'une div :

**Je suis le titre H1**

**Je suis le 1e titre H2**

Je suis le 2nd paragraphe

Je suis le 3e paragraphe

- ListeItem1
- ListeItem2
- ListeItem3
- ListeItem4
- ListeItem5

# Tester la présence d'attributs

hasAttribute() : Test la présence d'un attribut spécifique pour un élément.

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

/*Si p1 possède des attributs, hasAttributes() renvoie true et on exécute le code
 *de la condition*/
if(p1.hasAttributes()){
    vide.textContent = 'p1 possède des attributs';
}

//Si p1 possède un attribut id, hasAttribute() renvoie true
if(p1.hasAttribute('id')){
    vide.textContent += ' dont un attribut id';
}
```



# Récupérer/Définir un attribut

attributes : Liste contenant l'ensemble des attributs de l'élément

name : nom de l'attribut

value : valeur de l'attribut

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

if(p1.hasAttributes()){
  let attP1 = p1.attributes; // Liste des attributs de p1
  vide.innerHTML = 'Liste des attributs de p1 : <br>'
  //La propriété length renvoie le nombre d'attributs
  for(let i = 0; i < attP1.length; i++){
    vide.innerHTML += attP1[i].name + ' = ' + attP1[i].value + '<br>';
  }
}
```

# Récupérer un attribut

`getAttributeNames()` : renvoie les noms des attributs d'un élément en tableau

`getAttribute()` : Renvoi la valeur d'un attribut passé en argument.

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

//Si p1 possède des attributs...
if(p1.hasAttributes()){
  //On récupère leurs noms dans un tableau
  let attP1 = p1.getAttributeNames();
  vide.innerHTML = 'Attributs de p1 : <br>';

  //Pour chaque élément du tableau...
  for(nom of attP1){
    //On récupère la valeur associée au nom de l'attribut
    let valeur = p1.getAttribute(nom);
    //On affiche le nom et la valeur de l'attribut
    vide.innerHTML += nom + ' = ' + valeur + '<br>';
  }
}
```

# Définir un attribut

`setAttribute()` : Set la valeur d'un attribut passé en argument.

```
let p1 = document.querySelector('p');  
let p2 = document.getElementById('p2');  
let vide = document.getElementById('vide');  
  
p2.setAttribute('class', 'blue');  
vide.innerHTML += 'class : ' + p1.className + '<br>id : ' + p1.id;
```





# Supprimer un attribut

`removeAttribute()` : Supprime la valeur d'un attribut passé en argument.

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



# Exercice d'application : CRUD attributs

Créez un nouveau fichier HTML contenant le doctype, et le code ci dessous.

1) Ecrivez la fonction "changer\_style()" permettant de changer le style du paragraphe par :

- Une couleur blanche
- Un fond noir
- Une bordure de 5px

2) Définissez une classe css 'active' contenant les propriétés énoncées précédemment.

Modifier la fonction changer\_style() pour que celle-ci applique la classe "active" au paragraphe

```
<p id="parag1">Lorem ipsum dolor</p>  
<button onclick="changer_style()">Changer Style</button>
```



# Définition des évènements

Un évènement est une action qui se produit, et possède 2 caractéristiques :

- Action écoutable
- Action à laquelle on peut répondre

Exemple : Clic sur un bouton et affichage alert suite au clic



# Créer des gestionnaires d'évènements

La façon recommandée d'implémenter un gestionnaire d'évènements est d'utiliser la méthode `addEventListener()`

```
//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'});
```

# Supprimer des gestionnaires d'évènements

`removeEventListener()` : Supprime un gestionnaire d'évènement déclaré par `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);
```

# Modifier le comportement d'un évènement par défaut

preventDefault() : Permet de supprimer l'action par défaut d'un évènement

```
let envoi = document.getElementById('btn-envoi');  
envoi.addEventListener('click', testDonnees);  
  
function testDonnees(e){  
  /*Si (if...) les données ne remplissent pas certaines conditions, renvoie un  
  message et empêche l'action par défaut du clic = l'envoi du formulaire*/  
  alert('Envoi du formulaire bloqué');  
  e.preventDefault();  
}
```





# Exercice d'application : Gestion des évènements

Créez un nouveau fichier HTML contenant le doctype.  
Ajoutez une `<div id="output">` ainsi qu'un `<button>Click me</button>`.  
Au click sur le bouton, le texte sera modifié par "Vous m'avez cliqué".