programmation web en javascript

3/5

JS dans le navigateur : DOM et évènements

- tous les navigateurs disposent d'un interprète js embarqué
- charger les programmes js : requête http
- interagir avec le document : le dom
- réagir à des actions de l'utilisateur : évènements

Charger des programmes JS

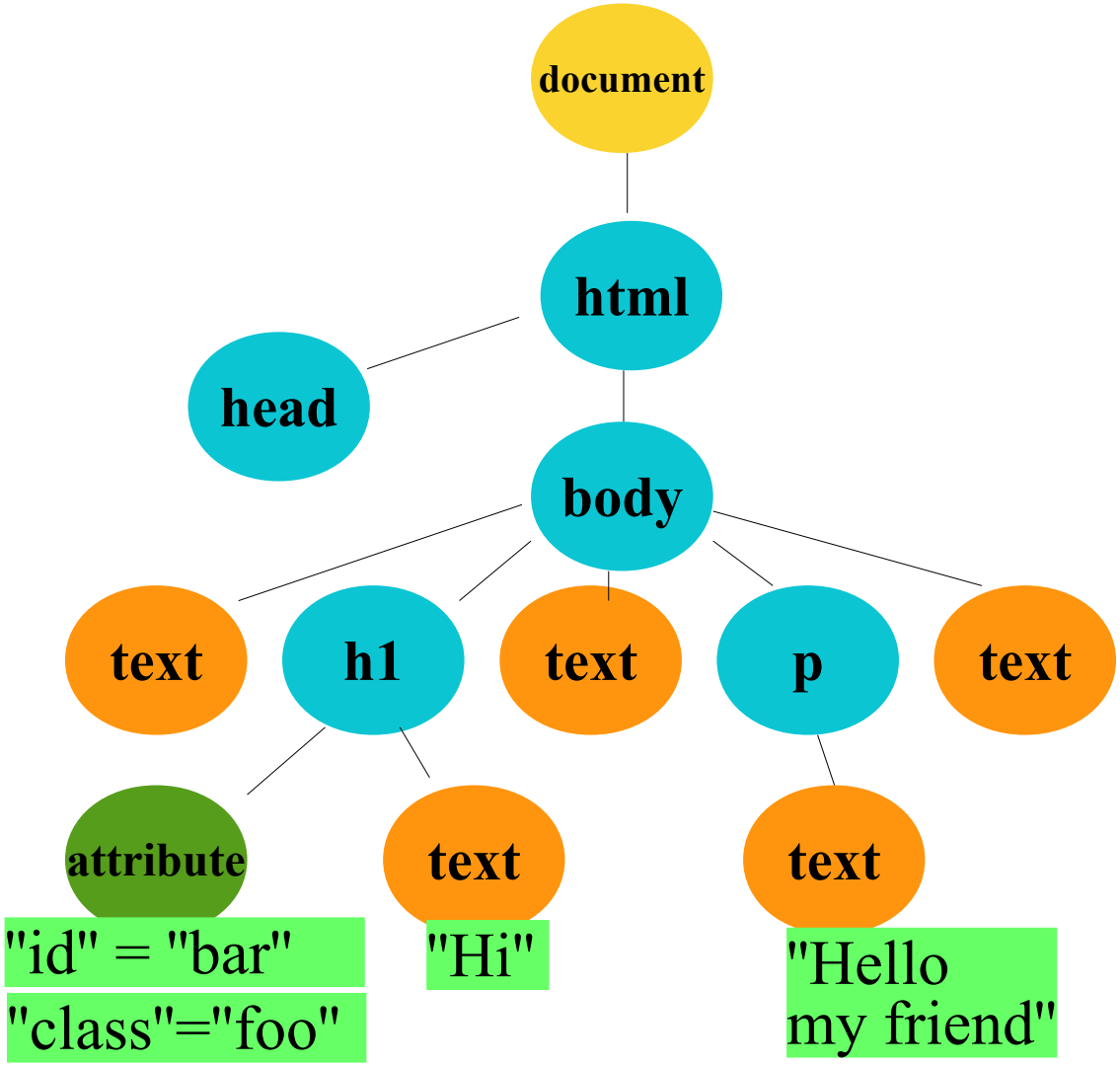
- inclure le code dans la page : la balise <script>
  - Dans la partie <head>
  - **Recommandé** : en fin de <body>
  - Permet de ne pas ralentir l'affichage de la page avec le téléchargement du JS

```
…
<body>
….
<script  src="monscript.js"> </script>
</body>
```

Le DOM

- dom : Document Object Model
- interface de manipulation d'un document arborescent, indépendante du langage
- chaque noeud de l'arbre = 1 objet
  - des propriétés : nom, attributs …
  - des méthodes pour manipuler l'arbre
- standard défini par W3C

```
<!DOCTYPE html>
<html>
<head> … </head>
<body>

<h1 id="bar" class="foo">
 Hi
</h1>

<p> Hello my friend
</p>

</body>
</html>
```



- 1 document = 1 arbre
- chaque nœud = 1 objet `node`

- noeud **`document`**
- noeud **`element`**
- noeud **`attribute`**
- noeud **`text`**

Les noeuds

- Les propriétés de **node**

Type de noeud	nodeType	nodeName	nodeValue	id	className
Element	1	Nom balise	null	identifiant	classe
Attribute	2	Nom attribut	valeur attribut	-	-
Text	3	#text	valeur texte	-	-
Document	9	#document	null	-	-

Interface avec le DOM

- javascript fournit 1 interface pour manipuler le dom :
- sélectionner et accéder à des éléments de l'arbre dom, naviguer dans le dom
- créer, insérer, supprimer, modifier des éléments dans le dom
- écouter des évènements produits par des éléments dans le dom

Sélectionner des nœuds dans le DOM

- `element.querySelector(sel)`
  - retourne le **1er élément** parmi les descendants du nœud sur lequel on l'invoque et qui correspond au sélecteur spécifié
- `element.querySelectorAll(sel)`
  - retourne **1 liste** contenant **tous** les éléments descendants du nœud sur lequel on l'invoque et qui correspondent au sélecteur spécifié
- où `element` peut être `document` ou un nœud quelconque
- et `sel` est un `CSS selector`

```html
<html>
<head> … </head>
<body>
  <section class="main">
     <p id="p1"> paragraphe 1 </p>
     <p id="p2"> paragraphe 2 </p>
     <p id="p3"> paragraphe 3 </p>
  </section>
</body>
</html>
```

```js
let p1 = document.querySelector("#p1");
console.log(p1.nodeName, p1.id);
// P p1

let paragraphes =
     document.querySelectorAll("p");
console.log(paragraphes[1].nodeName, paragraphes[1].id);
// P p2

let mains = document.querySelectorAll(".main");
console.log(mains[0].nodeName, mains[0].className);
// SECTION main
```

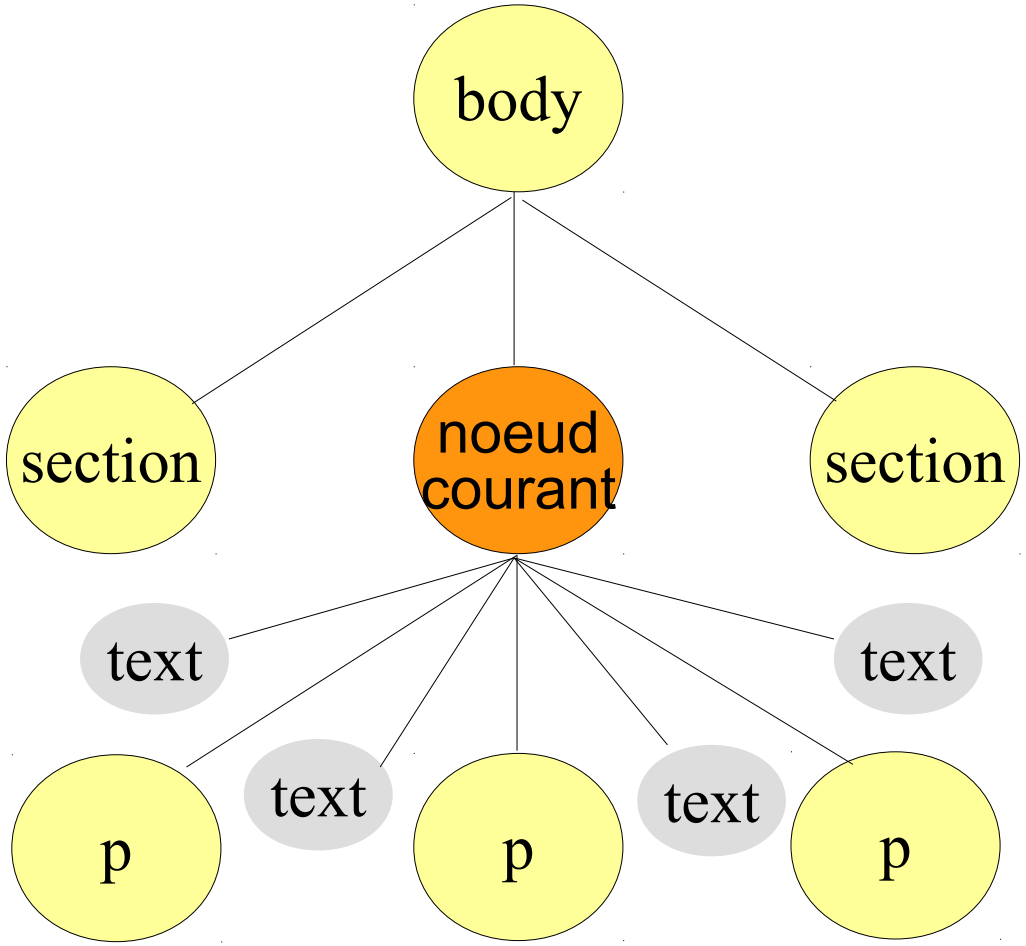
Sélectionner des nœuds dans le DOM

- `document.getElementById("id")`
  - retourne 1 nœud dont l'id est `"id"`
- `document.getElementsByTagName("tag")`
  - retourne les nœuds dont la balise est `"tag"`
- `document.getElementsByClassName("class")`
  - retourne les nœuds dont la classe est `"class"`

```html
<html>
<head> … </head>
<body>
  <section class="main">
     <p id="p1"> paragraphe 1 </p>
     <p id="p2"> paragraphe 2 </p>
     <p id="p3"> paragraphe 3 </p>
  </section>
</body>
</html>
```

```js
let p1 = document.getElementById("p1");
console.log(p1.nodeName, p1.id);
// P p1

let paragraphes =
     document.getElementsByTagName("p");
console.log(paragraphes[1].nodeName, paragraphes[1].id);
// P p2

let mains = document.getElementsByClassName("main");
console.log(mains[0].nodeName, mains[0].className);
// SECTION main
```

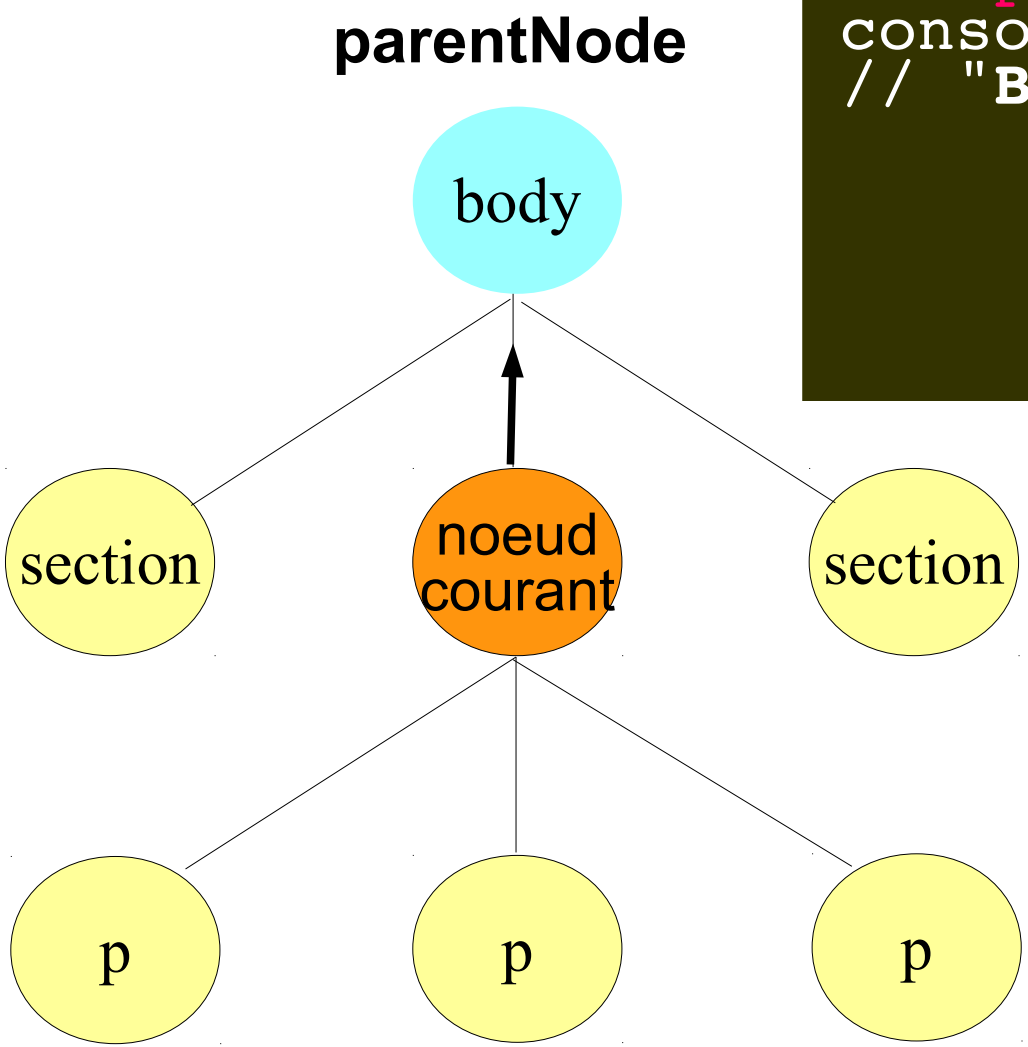
Se déplacer dans l'arbre




```
<html>
<head> … </head>
<body>
 <section> ... </section>
 <section id="main">
  <p> paragraphe 1 </p>
  <p> paragraphe 2 </p>
  <p> paragraphe 3 </p>
 </section>
 <section> ... </section>
</body> </html>
```

```
let s = document
     .querySelector("section#main") ;

console.log(s.nodeName);
// "SECTION"
```

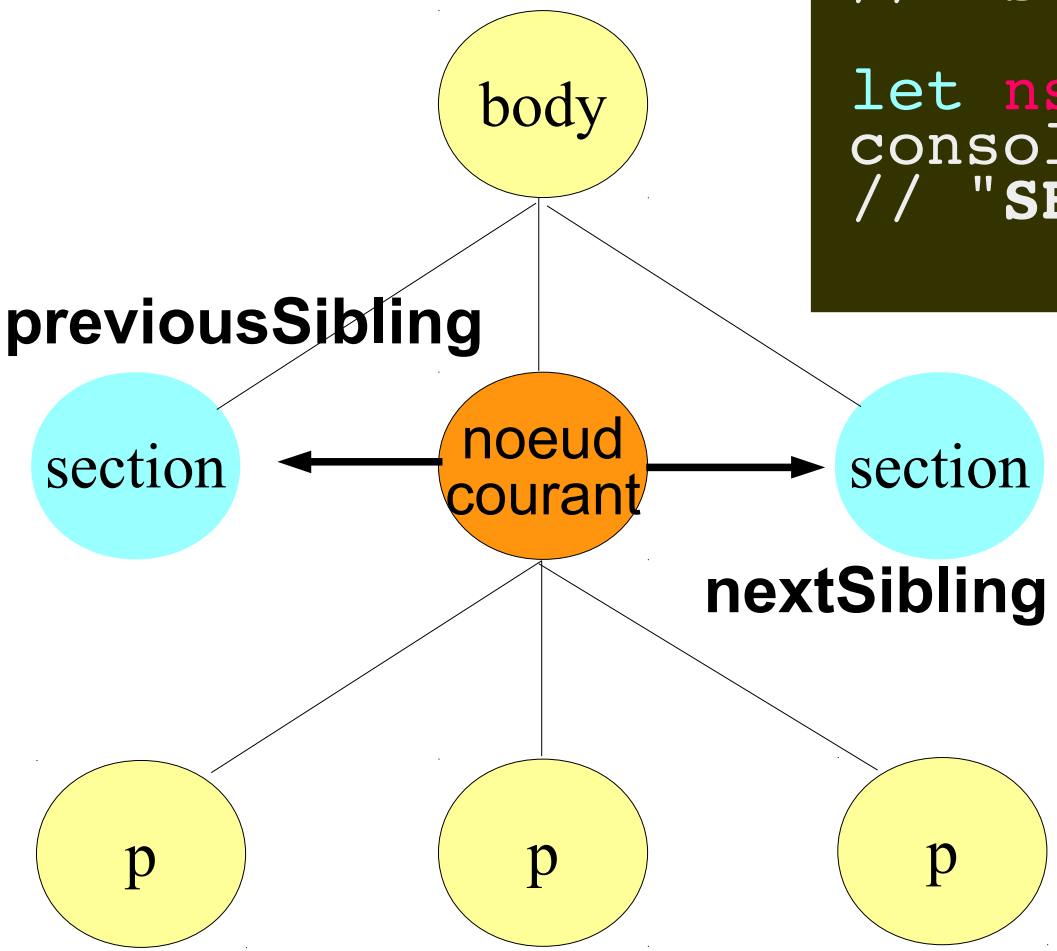
Se déplacer dans l'arbre




```
let s = document.
    querySelector("section#main") ;

let p = s.parentNode ;
console.log(p.nodeName);
// "BODY"
```

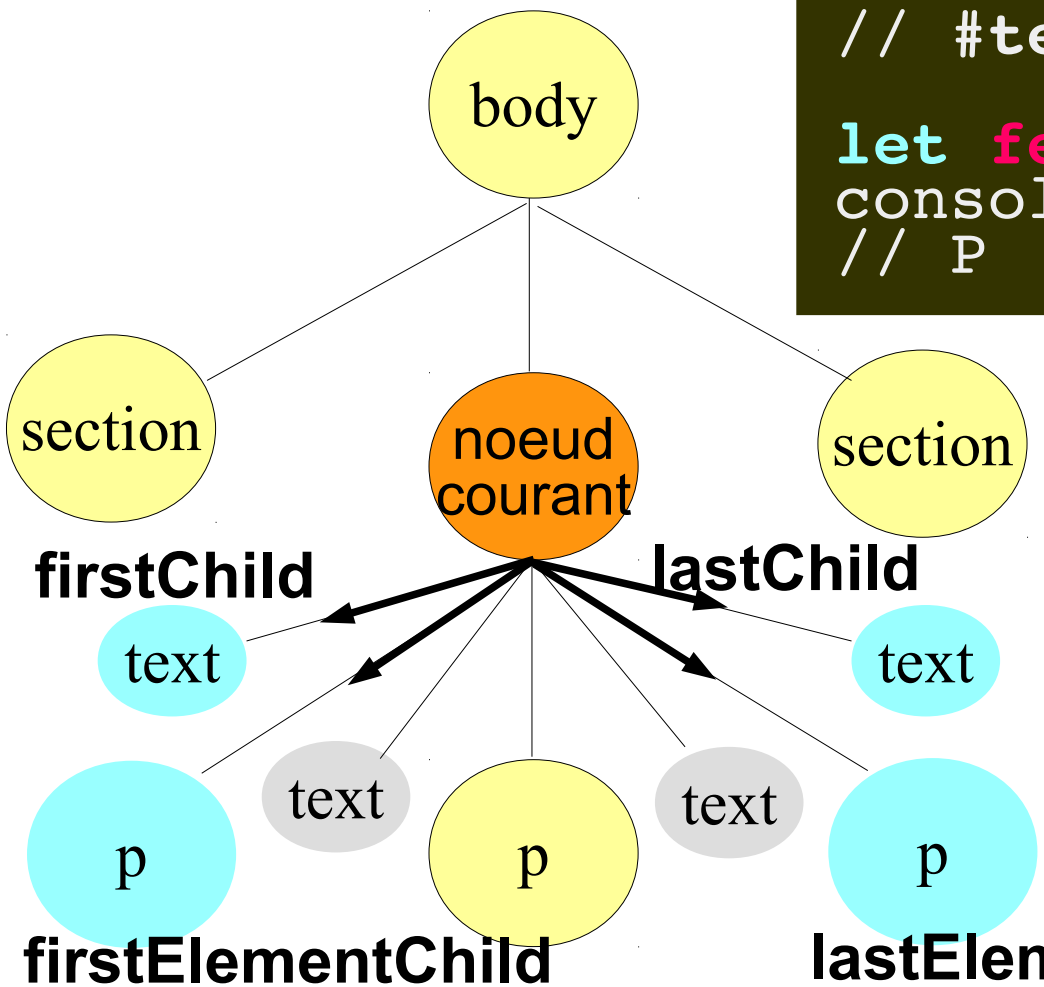
Se déplacer dans l'arbre

```
let s = document.
    querySelector("section#main") ;

let ps = s.previousSibling ;
console.log(ps.nodeName);
// "SECTION"

let ns = s.nextSibling ;
console.log(ns.nodeName);
// "SECTION"
```



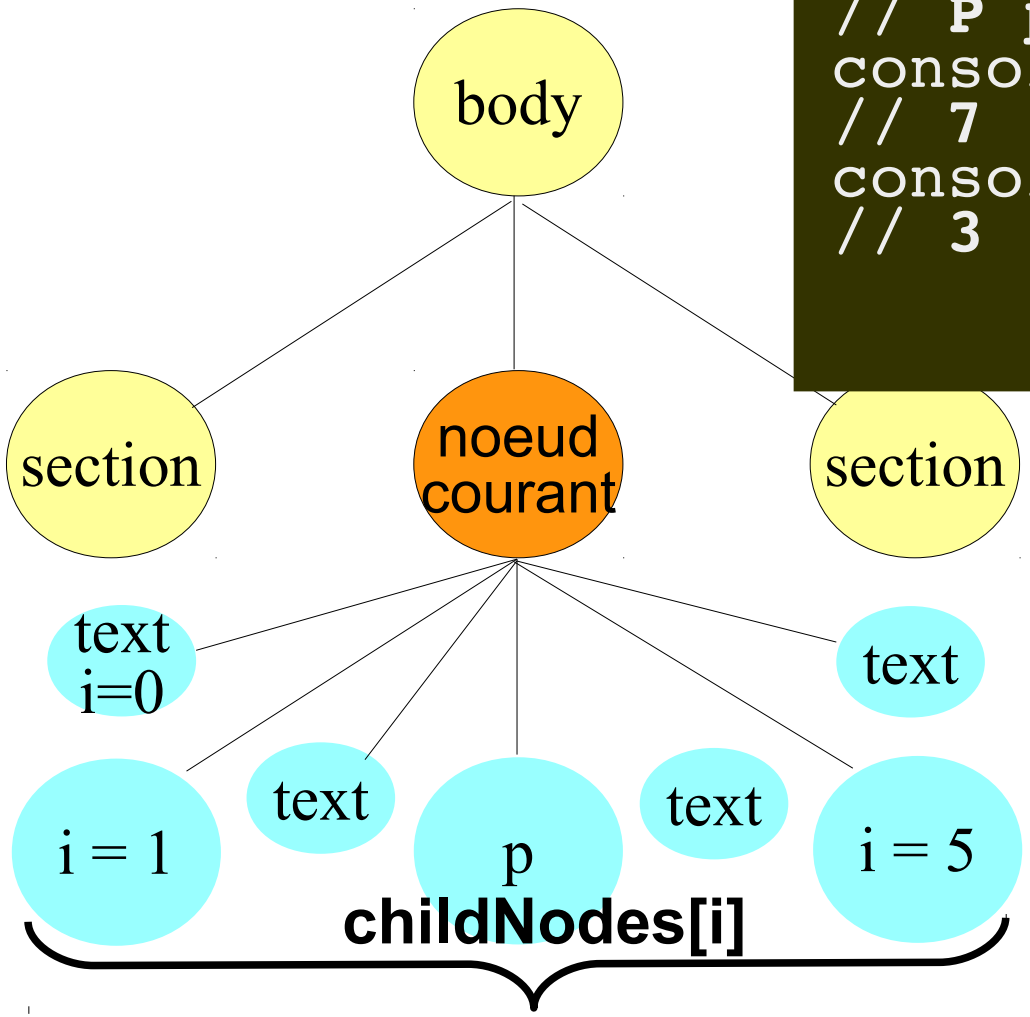
Se déplacer dans l'arbre

```
let fs = s.firstChild ,
console.log(fs.nodeName);
// #text

let ls = s.lastChild ,
console.log(ls.nodeName);
// #text

let fes = s.firstElementChild ,
console.log(fes.nodeName);
// P
```



Se déplacer dans l'arbre

```
let s = document.
    querySelector("section#main") ;

let lc =s.childNodes ;
console.log(lc[1].nodeName) ;
// P p1
console.log(lc.length) ;
// 7
console.log(s.childElementCount) ;
// 3
```



Modifier l'apparence du document

- on modifie l'apparence d'un document en ajoutant ou retirant des **classes css** sur 1 élément :
  - *elt*.`classList` : liste des classes portées par el*t* – propriété non modifiable directement
  - *elt*.`classList.add("foo")` : ajoute la classe "foo" à l'élément e*lt*
  - *elt*.`classList.remove("bar")` : retire la classe "bar" à l'élément e*lt*
  - *elt*.`classList.toggle("chu")` : ajoute/retire la classe "chu" à l'élément e*lt* si elle est absente/présente
  - *elt*.`classList.contains("man")` : retourne true si l'élément e*lt* porte la classe "man"

Modifier l'arbre dom

- créer des nœuds, puis
- les insérer dans l'arbre
- supprimer des nœuds de l'arbre

- **créer des nœuds :**
  - `document.createElement("b")` : crée 1 noeud **B**
  - `document.createTextNode('texte')` : crée 1 noeud Text avec le texte 'texte'
  - *node*`.setAttribute(att,val)` : affecte 1 valeur à 1 attribut du noeud element *node*
- **insérer des nœuds dans le dom :**
  - *node*`.appendChild(n)` : insère le noeud **n** comme dernier enfant
  - *node*`.insertBefore(n,nb)` : insère le noeud **n** avant **nb** dans les enfants de *node*
  - *node*`.replaceChild(n,nb)` : remplace **nb** par **n** dans les enfants de *node*
  - *node*`.removeChild(n)` : supprime le fils **n**

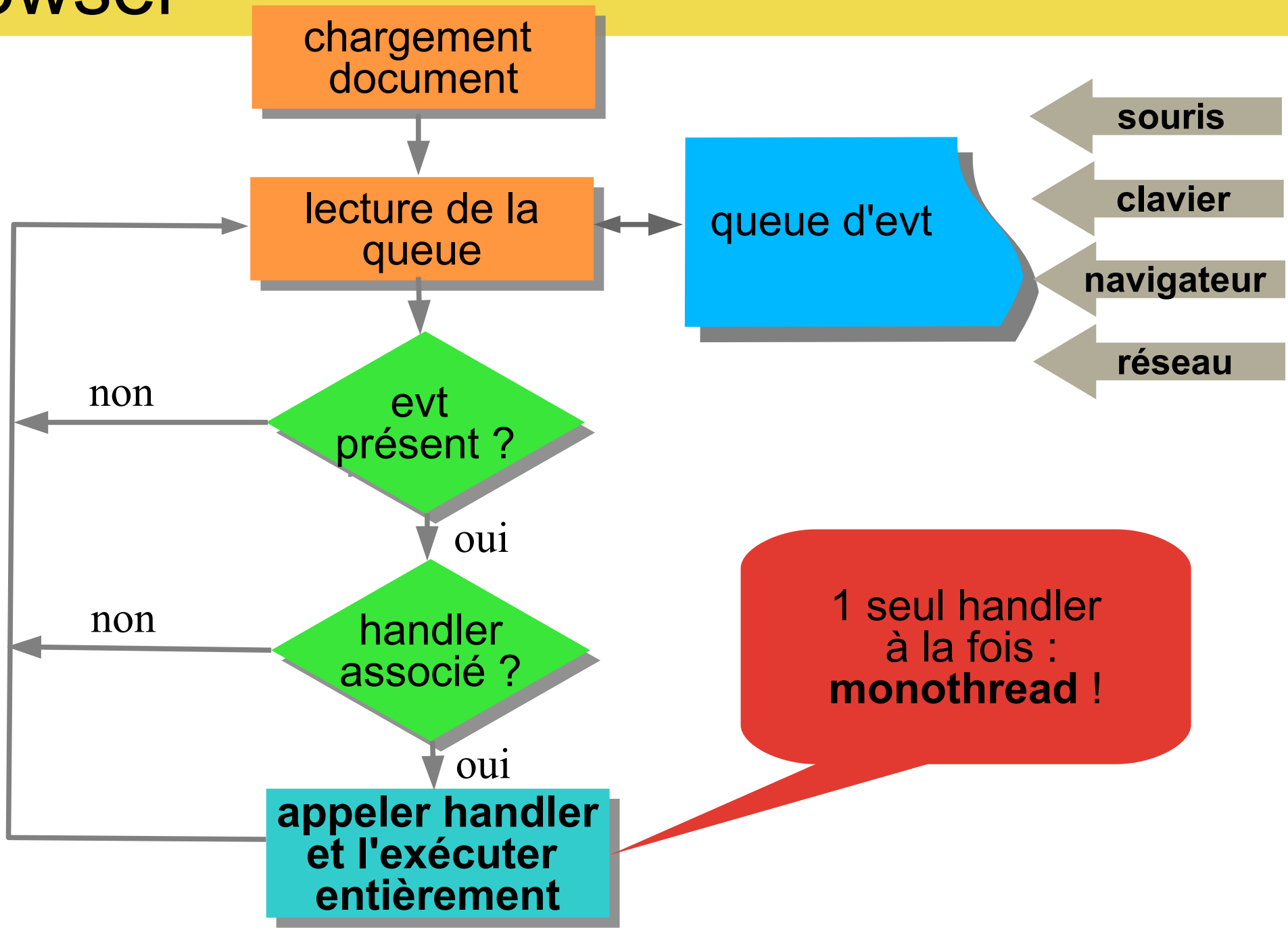
```html
<html>
<head> … </head>
<body>
  <section class="main">
     <p id="p1"> paragraphe 1 </p>
     <p id="p2"> paragraphe 2 </p>
     <p id="p3"> paragraphe 3 </p>
  </section>
</body>
</html>
```

```javascript
// ajouter un paragraphe
let s = document.querySelector("section.main");
let p = document.createElement("P");
p.setAttribute("id", "p4");

let t = document.createTextNode("paragraphe 4");
p.appendChild(t);

s.appendChild(p);

// insérer 1 ligne entre les paragraphes
let p3 =  document.querySelector("section>p#p3");

s.insertBefore(document.createElement("HR"), p3);
```

Les évènements

- les objets du dom génèrent des évènements correspondant souvent à des actions de l'utilisateur sur l'interface (ils sont *observables*)
- pour réagir à ces actions : on lie une fonction à 1 évènement
  - en js, les ***observers*** sont des fonctions, nommées aussi *handler*
- une application js dans le navigateur est une application conduite par des évènements
- la boucle d'évènements est réalisée par l'interprète js

La boucle d'évènements : JS dans le browser



Les évènements

- les évènements sont produits par les nœuds du dom

evt	nodeType	Action détectée
load	document,frameset	chargement complet du dom
click	element	click de souris
mousedown	element	bouton de souris enfoncé
mouseup	element	bouton de souris relaché
mouseover	element	passage du pointeur au dessus
keypress	element	touche clavier pressée
blur	element	perte du focus (utile dans 1 form)
change	element	modification de la valeur (dans 1 saisie)
resize	document, element	changement de taille
scroll	document, element	utilisation du scroll
focus	element	acquisition du focus

Enregistrement de handler

- on enregistre 1 listener sur 1 nœud pour 1 évènement
- on peut enregistrer plusieurs listeners sur le même nœud pour 1 même évènement

- `node.addEventListener("evt", handler)`
  - `"evt"` : l'évènement concerné
  - `handler` : une fonction javascript recevant un objet *`event`* en paramètre
- il faut être sûr que le dom est **entièrement chargé** pour enregistrer un handler

```
<html>
<head> … </head>
  <body>
    <a id="a1" href="#"> vroum</a>
    <a id="a2" href="#"> weezz</a>
  </body>
</html>
```

- déclarer des handlers :

```
let a1 = document.querySelector("#a1");
let a2 = document.querySelector("#a2");

a1.addEventListener("click", (event) => {
      alert("vrrooouuuumm");
});

a2.addEventListener("mouseover", (event) => {
      alert ("weeeeezzzz");
});
```

Enregistrement de handlers

- Il est important d'être sûr que le **DOM** est entièrement **chargé** avant de sélectionner des nœuds et de déclarer des handlers :

```
window.addEventListener("load", () => {

      let a1 = document.querySelector("#a1");
      let a2 = document.querySelector("#a2");

      a1.addEventListener("click", (event) => {
            alert("vrrooouuuumm");
      });

      a2.addEventListener("mouseover", (event) => {
            alert("weeeeezzzz");
      });
});
```

Programmation des handlers

- un handler reçoit comme seul argument un objet *`event`*
- selon le type de l'évènement, on a accès à différentes propriétés/méthodes

- pour annuler l'évt : *`event`*`.preventDefault()`

- pour accéder à la cible (nœud du DOM ayant reçu l'événement) :
  - *`event`*`.target :` dans tous les cas
  - `this :` dans les handlers définis avec `function()`

```
document.querySelector("#the-btn")
    .addEventListener("click", (e) => {
            e.target.classList.toggle("btn-green"); });


document.querySelector("a.no-reload")
    .addEventListener("click", (e)
                    => { e.preventDefault(); });


document.addEventListener("mousedown", (e) => {
            console.log("x:", e.clientX, "y:", e.clientY);
});


document.addEventListener("keypress", (e) => {
     console.log("key: ", e.key,
                    "keycode: ", e.keyCode);
});
```

Autre méthode d'enregistrement de handler

- donner une valeur à la propriété `on<evt>` d'un nœud du DOM
  - 'load' → onload , 'click' → onclick
- Lorsque l'evt se produit cette valeur est utilisée comme handler d'evt
  - *node*`.onclick = () => { … }`
  - *node*`.onmouseover = animateButton`
  - `window.onload = () => { … }`
- ATTENTION : ne permet pas d'enregistrer plusieurs handlers sur le même évènement

Autre méthode d'enregistrement de handler

- associer un appel javascript à 1 évènement dans le document html : **mauvaise pratique**

```
<body onload="alert('document chargé') ;">
…
<form>
<input type="text" name="id"
                  onclick="verifier_nom() ;">
```