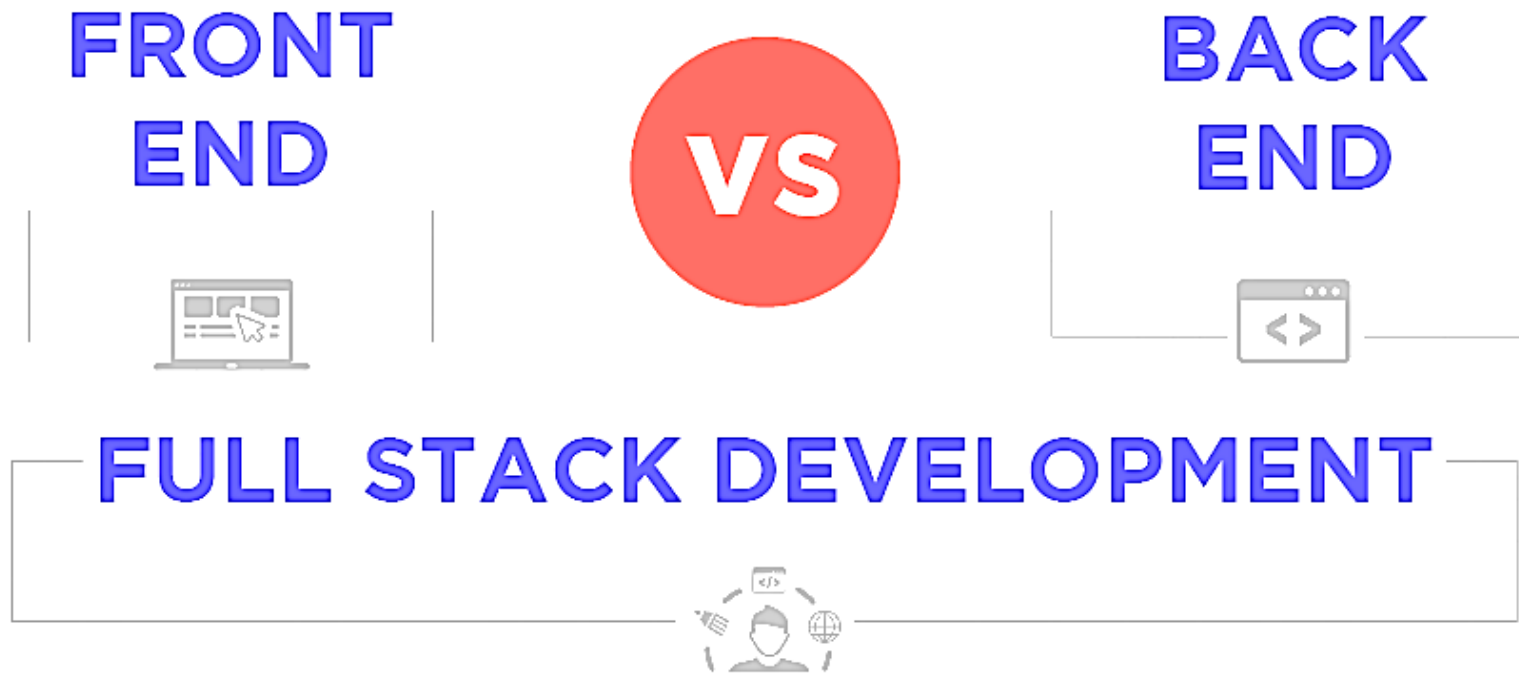


Programmation Web Front-end

HTML, CSS, JavaScript, Référencement Naturel SEO

Leçon 4 : JavaScript

Rappel: Développement Full Stack



La définition et l'usage du JavaScript

JavaScript:

- Langage populaire et open source
- Le JavaScript permet aux développeurs de travailler avec un seul langage de programmation, que ce soit pour le développement **front-end** (l'application ou le site Web - le client) ou le développement **back-end** (bases de données, API, authentification, etc. - le serveur).
- Les développeurs utilisent **un type de JavaScript appelé Node.js** pour le travail **back-end**. Node.js permet d'exécuter un code JavaScript à l'extérieur d'un navigateur web.

La définition et l'usage du JavaScript

- Pourquoi apprendre le JavaScript?
- Le JavaScript est le langage de préférence pour interagir avec le HTML permettant ainsi d'**apporter du dynamisme à l'intérieur des pages web**.
- C'est le meilleur pour faire ses premiers pas et apprendre la programmation pour plusieurs raisons. La **syntaxe est assez simple et intuitive** et très proche de l'anglais.

Le javascript, ça sert à quoi ?

- Contrôler des données saisies dans des formulaires HTML
- Vérifier ces données
- Rendre un site dynamique vivant
- Interagir avec le document HTML
- Créer des infobulles
- Faire défiler des images
- Afficher / masquer du texte
- Modifier du css
- Créer des événements, des diaporamas
- ...

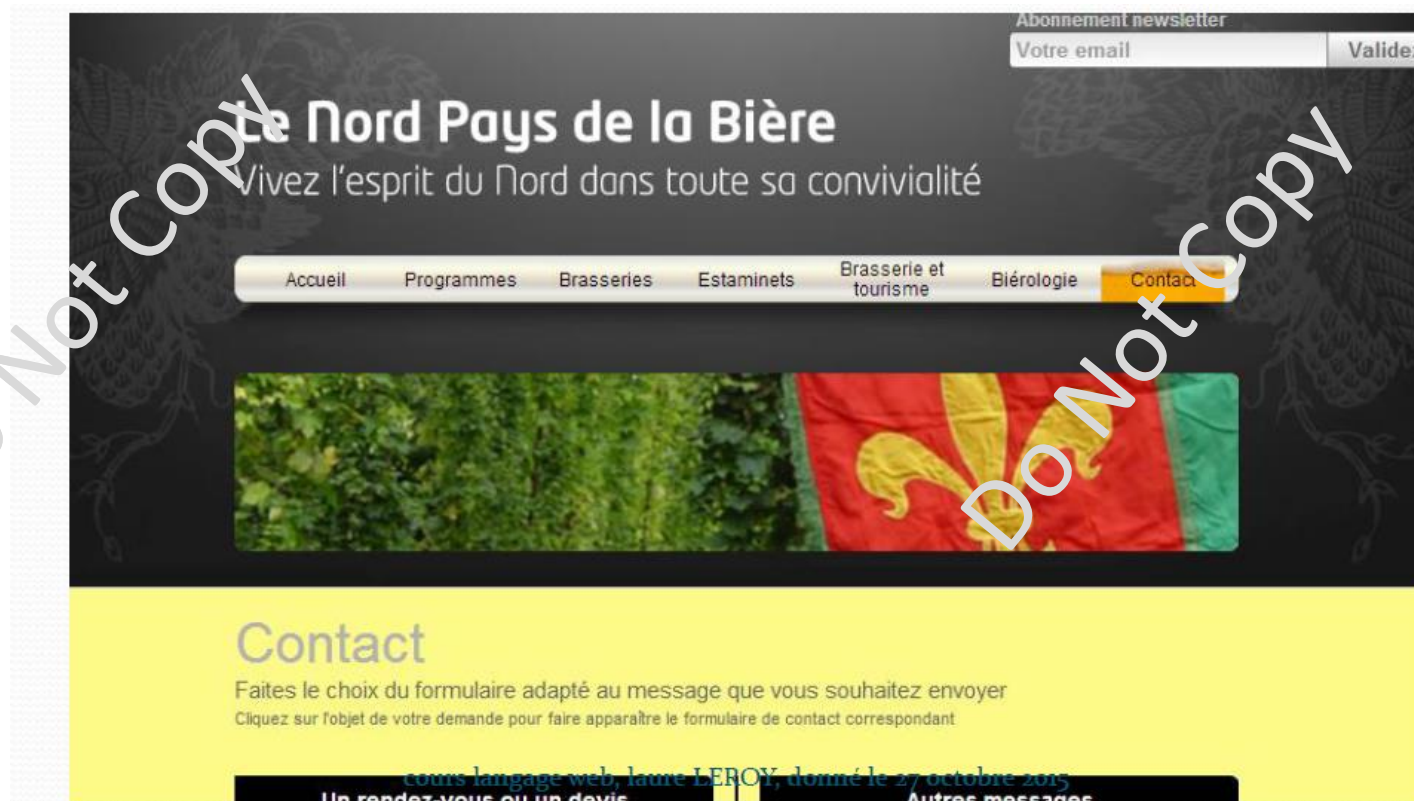
Le javascript, ça sert à quoi ?

- Exemple: Un texte qui s'adapte aux images



Le javascript, ça sert à quoi ?

- Exemple: Défilement d'images



Le javascript, ça sert à quoi ?

- Exemple: Slider



Le javascript, ça sert à quoi ?

- Exemple: Formulaire

[[Liste des rédacteurs](#) | [Liste des mots-clés](#) | [Tableau chronologique](#)]

auteur	<input type="text"/>	<div>dans n'importe quel champ</div> <input type="text"/> <div>textes numérisés</div> <input type="radio"/> non <input type="radio"/> oui
titre	<input type="text"/>	
domaine	<input type="text"/>	
période	<input type="text"/>	
langue	<input type="text"/>	
rédacteur	<input type="text"/>	
mot-clé	<input type="text"/>	

☒ par auteurs ☐ par titres ☐ par codes

[cours langage web, laure LEROY, donné le 27 octobre 2015](#)

Le javascript, ça sert à quoi ?

- Exemple: Drag & Drop



Le javascript, ça sert à quoi ?

- Exemple: Les jeux des petits chevaux



Avantages et Inconvénients

- **Avantages**

- Supporté (par défaut) par les principaux navigateurs, ne nécessite pas de plug in particulier.
- Accès aux objets contenus dans un document HTML (+ possibilité de les manipuler relativement facilement).
- Possibilité de mettre en place des animations.

- **Inconvénients**

- Les utilisateurs peuvent empêcher l'exécution de code JavaScript, souvent en raison des erreurs générées par les scripts, ou encore en raison de la nature de l'interaction (apparition de nouvelles fenêtres, ...).
- Lenteur d'exécution des scripts, ainsi que pour les scripts complexes, un certain délai avant le démarrage.

La définition et l'usage du JavaScript

- Déclencher le code JavaScript depuis le document HTML:
 - Dans un navigateur, JavaScript ne fait rien « tout seul ».
 - **Le code JavaScript a besoin d'être lancé depuis les pages web HTML.**
 - Pour appeler du code JavaScript depuis votre document HTML, vous aurez besoin de l'élément
`<script>`

Premier Exemple d'un code HTML/CSS/JS

```
1 <html>
2   <head>
3     <title>Le parcours NET</title>
4     <style type="text/css">
5       body {
6         color: purple;
7         background-color: #d8da3d }
8     </style>
9   </head>
10  <body>
11    <h1>Université Paris 8</h1>
12    <h2>M1 NET</h2>
13    <p>Bienvenue sur la page du Master 1 Numérique : Enjeux, Technologies!
14    <br>
15    <script>
16      document.write("bonjour!");
17
18      alert("c'est un \"message\" d'alerte");
19    </script>
20  </body>
21 </html>
```

Code CSS

Code JavaScript



JavaScript

- Pour déclencher le code JavaScript depuis le document HTML :
 - Dans un navigateur, JavaScript ne fait rien « tout seul ».
 - Il a besoin d'être lancé depuis les pages web HTML.
- Pour appeler du code JavaScript depuis votre document HTML :
 - vous aurez besoin de l'élément `<script>`

Pour qu'un navigateur puisse différencier le code en Javascript du reste du document HTML, on l'encapsule dans une balise spécifique :

```
<script language="Javascript">  
...  
</script>
```

`<script language="javascript">` : utilisé dans les anciens navigateurs, est devenu obsolète

`<script type="text/javascript">` : Standard HTML 4

En HTML 5, le paramètre type is optionnel (text/javascript est inséré par défaut), donc on peut tout simplement écrire `<script>`.

JavaScript

- Exemple d'un élément JS script, au sein d'une page HTML :

```
<html>
  <head>
    <meta charset="utf-8">
    <title>TP 1</title>
    <link rel="stylesheet" href="monstyle.css">
    <script>
      document.addEventListener('DOMContentLoaded', function(){

        let bonjour = document.getElementById('b1');
        bonjour.addEventListener('click', alerte);
        function alerte(){
          alert('Bonjour SRT3 !');
        }
      });
    </script>
  </head>
  <body>
    <h1> Premier exemple Javascript </h1>
    <button id = 'b1'> Dites Bonjour </button>

  </body>
</html>
```

JavaScript

- On peut ajouter du JS directement dans la balise ouvrante d'un élément HTML :

```
Exemple1Javascript.html x
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>TP 1</title>
6   </head>
7   <body>
8     <h1> Premier exemple Javascript </h1>
9     <button onclick = "alert('Bonjour SRT3 ! ')"> Dites Bonjour </button>
10
11   </body>
12 </html>
```

JavaScript

- Dans un fichier séparé contenant exclusivement du JavaScript et portant l'extension .js :

```
Exemple3Javascript.html x ExempleJs.js x
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>TP 1</title>
6     <link rel="stylesheet" href="monstyle.css">
7     <script src='ExempleJs.js' async ></script>
8
9   </head>
10  <body>
11    <h1> Premier exemple Javascript </h1>
12    <button id = 'b1'> Dites Bonjour </button>
13
14  </body>
15 </html>
16
```

```
Exemple3Javascript.html x ExempleJs.js
1 let bonjour = document.getElementById('b1');
2
3 bonjour.addEventListener('click', alerte);
4
5 function alerte(){
6   alert('Bonjour SRT3 !');
7 }
```

La syntaxe

- JavaScript est sensible à la casse.
- En JavaScript, les instructions sont séparées par des ;
- S'il y'a plusieurs déclarations sur une seule ligne, alors elles doivent être séparées par un ;
- Les commentaires :

```
// un commentaire sur une ligne

/* un commentaire plus
   long sur plusieurs lignes
*/

/* Par contre on ne peut pas /* imbriquer des commentaires */ SyntaxError */
```

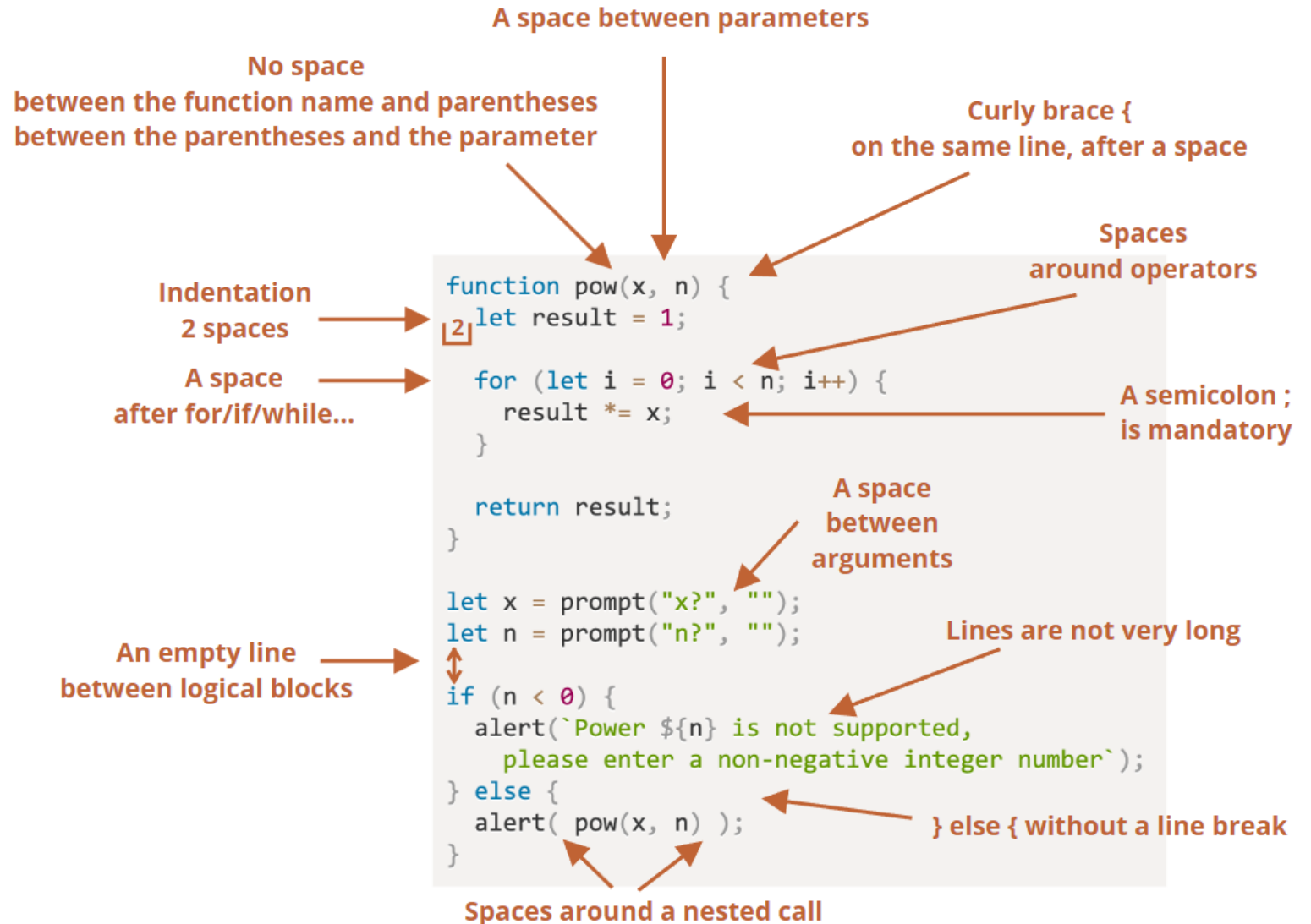
Bon style de codage

- Notre code doit être aussi propre et lisible que possible.
- C'est en fait un art de la programmation – prendre une tâche complexe et la coder de manière correcte et lisible par l'homme. Un bon style de code aide grandement à cela.
- Consulter :
<https://fr.javascript.info/coding-style>

Bon style de codage

- Voici un aide-mémoire avec quelques règles suggérées:

NB: Ce sont des préférences de style;
Pas de règles strictes



Bon style de codage : Code propre et lisible

- Accolades :

Dans la plupart des projets JavaScript, les accolades sont écrites sur la même ligne que le mot clé correspondant, et non sur la nouvelle ligne, dans un style dit «égyptien». Il y a aussi un espace avant un crochet d'ouverture.

Comme ceci :

```
1  if (condition) {  
2    // fait ceci  
3    // ...et cela  
4    // ...et cela  
5  }
```

Bon style de codage : Code propre et lisible

- Accolades :

Une construction sur une seule ligne, comme `if (condition) doSomething()`, est un cas important. Devrions-nous utiliser des accolades ?

Voici les variantes annotées pour que vous puissiez juger de leur lisibilité :

1. 🤔 Les débutants font parfois cela. C'est une mauvaise pratique! Les accolades ne sont pas nécessaires :

```
1 if (n < 0) {alert(`Power ${n} is not supported`);}
```

2. 😬 Lorsque vous n'utilisez pas d'accolades, évitez de passer pas à la ligne, il est facile de se tromper! :

```
1 if (n < 0)
2   alert(`Power ${n} is not supported`);
```

3. 😊 Ne pas utiliser d'accolade sur une seule ligne, est acceptable tant que cela reste court :

```
1 if (n < 0) alert(`Power ${n} is not supported`);
```

4. 😊 Voici une bonne manière de faire :

```
1 if (n < 0) {
2   alert(`Power ${n} is not supported`);
3 }
```

Pour un code très court, une ligne est autorisée, par exemple. `if (cond) return null`. Mais la variante numéro 4 est généralement plus lisible.

Bon style de codage : Code propre et lisible

- Longue de la ligne :

Personne n'aime lire une longue ligne horizontale de code. La meilleure pratique est de les scinder.

Par exemple :

```
1 // les guillemets backtick ` permettent de scinder la chaîne de caractères en plusieu
2 let str = `
3   ECMA International's TC39 is a group of JavaScript developers,
4   implementers, academics, and more, collaborating with the community
5   to maintain and evolve the definition of JavaScript.
6 `;
```

Et pour les déclarations `if` :

```
1 if (
2   id === 123 &&
3   moonPhase === 'Waning Gibbous' &&
4   zodiacSign === 'Libra'
5 ) {
6   letTheSorceryBegin();
7 }
```

La longueur de ligne maximale est convenue au niveau de l'équipe. C'est généralement 80 ou 120 caractères.

Bon style de codage : Code propre et lisible

- Indentation :

- **Un retrait horizontal : 2(4) espaces.**

Une indentation horizontale est faite en utilisant 2 ou 4 espaces ou le symbole horizontal de tabulation (key `Tab`). Lequel choisir est une vieille guerre sainte. Les espaces sont plus communs de nos jours.

Un des avantages des espaces sur les tabulations est qu'ils permettent des configurations de retrait plus flexibles que le symbole tabulation.

Par exemple, nous pouvons aligner les arguments avec le crochet d'ouverture, comme ceci :

```
1 show(parameters,  
2     aligned, // 5 espaces à gauche  
3     one,  
4     after,  
5     another  
6 ) {  
7     // ...  
8 }
```

Bon style de codage : Code propre et lisible

- Indentation :

- **Un retrait vertical: lignes vides pour fractionner le code en blocs logiques.**

Même une seule fonction peut souvent être divisée en blocs logiques. Dans l'exemple ci-dessous, l'initialisation des variables, la boucle principale et le retour du résultat sont fractionnés verticalement :

```
1 function pow(x, n) {  
2   let result = 1;  
3   //           <--  
4   for (let i = 0; i < n; i++) {  
5     result *= x;  
6   }  
7   //           <--  
8   return result;  
9 }
```

Insérez une nouvelle ligne où cela aide à rendre le code plus lisible. Il ne devrait pas y avoir plus de neuf lignes de code sans indentation verticale.

Bon style de codage : Code propre et lisible

- Point-virgule :

- Un point-virgule doit être présent après chaque déclaration. Même si cela pourrait éventuellement être ignoré.
- Il y a des langages où le point-virgule est vraiment optionnel. Il est donc rarement utilisé. Mais dans JavaScript, il y a peu de cas où un saut de ligne n'est parfois pas interprété comme un point-virgule. Cela laisse place à des erreurs de programmation.
- Si vous êtes un programmeur JavaScript expérimenté, vous pouvez choisir un style de code sans point-virgule comme [StandardJS](#). Autrement, **il est préférable d'utiliser des points-virgules pour éviter les pièges possibles**. La majorité des développeurs mettent des points-virgules.

Bon style de codage : Code propre et lisible

- Niveaux d'imbrication :

Il ne devrait pas y avoir trop de niveaux d'imbrication.

Par exemple, dans une boucle, c'est parfois une bonne idée d'utiliser la directive "continue" pour éviter une imbrication supplémentaire.

Par exemple, au lieu d'ajouter un `if` imbriqué conditionnel comme ceci :

```
1 for (let i = 0; i < 10; i++) {  
2   if (cond) {  
3     ... // <- un autre niveau d'imbrication  
4   }  
5 }
```

Nous pouvons écrire :

```
1 for (let i = 0; i < 10; i++) {  
2   if (!cond) continue;  
3   ... // <- pas de niveau d'imbrication supplémentaire  
4 }
```

Une chose similaire peut être faite avec `if/else` et `return`.

Bon style de codage : Code propre et lisible

- Niveaux d'imbrication :

Par exemple, les deux constructions ci-dessous sont identiques.

```
1 function pow(x, n) {  
2   if (n < 0) {  
3     alert("Negative 'n' not supported");  
4   } else {  
5     let result = 1;  
6  
7     for (let i = 0; i < n; i++) {  
8       result *= x;  
9     }  
10  
11    return result;  
12  }  
13 }
```

```
1 function pow(x, n) {  
2   if (n < 0) {  
3     alert("Negative 'n' not supported");  
4     return;  
5   }  
6  
7   let result = 1;  
8  
9   for (let i = 0; i < n; i++) {  
10    result *= x;  
11  }  
12  
13  return result;  
14 }
```

Le second est plus lisible, parce que le "cas marginal" de `n < 0` est traité tôt. Une fois la vérification effectuée, nous pouvons passer au flux de code "principal" sans avoir besoin d'imbrication supplémentaire.

Bon style de codage : Code propre et lisible

- Placement de fonctions :

Si vous écrivez plusieurs fonctions "helper" (auxiliaires) et le code pour les utiliser, il existe trois façons de les placer.

1. Déclarez les fonctions *au-dessus* du code qui les utilise :

```
1 // fonctions declarations
2 function createElement() {
3   ...
4 }
5
6 function setHandler(elem) {
7   ...
8 }
9
10 function walkAround() {
11   ...
12 }
13
14 // le code qui les utilise
15 let elem = createElement();
16 setHandler(elem);
17 walkAround();
```

2. Le code d'abord, puis les fonctions

```
1 // le code qui utilise les fonctions
2 let elem = createElement();
3 setHandler(elem);
4 walkAround();
5
6 // --- fonctions helper ---
7
8 function createElement() {
9   ...
10 }
11
12 function setHandler(elem) {
13   ...
14 }
15
16 function walkAround() {
17   ...
18 }
```

La plupart du temps, la deuxième variante est préférée.

C'est parce qu'en lisant du code, nous voulons d'abord savoir ce qu'il fait. Si le code commence en premier, il devient clair dès le début. Ensuite, nous n'aurons peut-être pas besoin de lire les fonctions du tout, surtout si leur nom décrit ce qu'elles font réellement.

Bon style de codage : Code propre et lisible

- Guides de style :

- Il existe de nombreux guides existants à choisir.

Par exemple :

- [Google JavaScript Style Guide](#)
 - [Airbnb JavaScript Style Guide](#)
 - [Idiomatic.JS](#)
 - [StandardJS](#)
 - (il y en a plus)

Rappel: Ce sont des préférences de style; Pas de règles strictes

Les bases Javascript

Les opérateurs : +, -, /, *, %, == (valeurs), === (valeurs et types), !=(valeur), !==(valeurs ou types), <, >, etc

```
var x = 1;
var y = 3;
var z = x+y; //Addition
var t = x-z; //Soustraction
x += 1; //incrément de 1
```

Concaténation

```
var mois = 4;
var jour = 'Novembre';
var DateTP1 = mois + jour + '2020';
```

Affichage

```
var x = 1;
var y = 3;
alert('x =' + x + '\n y = ' + y);
alert(`x = ${x}
      y =  ${y}
      Leur somme vaut ${x+y}`);
```


Exercice 1 – Simple structuration html

NB: On va ajouter du JS dans les exercices suivants

//Exercice 1 : Simple structuration html

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Premiers algorithmes en JavaScript</title>
5
6    </head>
7    <body>
8
9      <h1>Cours: Introduction à la programmation </h1>
10     <h2>TP1: Base de programmation HTML et JavaScript </h2>
11
12
13
14
15   </body>
16 </html>
```

Exercice 2 – Inclusion d'un script JS pour affichage dans le document HTML

```
// Exercice 2: Premier affichage //écrire dans le document HTML
document.write('<br>Bonjour !');
```

```
<head>
  <title>Premiers algorithmes en JavaScript</title>
</head>
<body>

  <h1>Cours: Introduction à la programmation </h1>
  <h2>TP1: Base de programmation HTML et JavaScript </h2>

  <h1>Algorithmes en JavaScript:</h1>

  J'essaye le javascript...

  <script>
    /* Premier affichage */
    document.write('<br>Bonjour !');
  </script>

</body>
</html>
```

- Inclusion du code JavaScript directement dans la page

Exercice 3 – Affichage d'un message d'alerte

```
// Exercice 3 //écrire dans un message d'alerte  
alert("Voici un message d\'alerte!");
```

Exercice 4 – Première fonction simple

```
// Exercice 4 //créer et appeler une fonction d'affichage

//créer une fonction bonjour qui affiche "Bonjour les amis!" dans la page web
function bonjour()
{
    document.write('<br>Bonjour les amis!');
}

//appeler la fonction bonjour()
bonjour();
```

La déclaration de variables

La déclaration peut se faire de deux façons :

- soit **de façon explicite**, en faisant précéder la variable du mot clé **var** qui permet d'indiquer de façon rigoureuse qu'il s'agit d'une variable :
`var chaine= "bonjour"`
- soit **de façon implicite**, en laissant le navigateur déterminer qu'il s'agit d'une déclaration de variable.

Pour déclarer implicitement une variable, il suffit d'écrire le nom de la variable suivie du caractère = et de la valeur à affecter :

`chaine= "bonjour"`

- Même si une déclaration implicite est tout à fait reconnue par le navigateur, il est plus rigoureux de déclarer les variables de façon explicite avec le mot var.

Exercice 5 – Création de variables globales

```
// Exercice 5
/* Ecrire un code qui additionne deux valeurs
et affiche le résultat dans la console de JavaScript */
var a = 5;
var b = 6;
var c = a + b;
console.log (c);
```

Exercice 6 – Saisie de données par l'utilisateur

```
//Exercice 6
/* Utilise la fonction prompt qui permet de récupérer une valeur entrée par l'utilisateur.
Puis affiche cette valeur dans la page web sur une nouvelle ligne */
valeur = prompt("Entrez une valeur :");

document.write("<br>")
document.write(valeur);
```

La déclaration de variables globales, locales et constantes

- Le mot-clé **let** permet de déclarer des variables dont la portée est limitée à celle du bloc dans lequel elles sont déclarées.
 - un bloc, c'est simplement l'ensemble des instructions comprises entre deux accolades "{...}"
 - un bloc d'instruction se trouve souvent après un if, else, for, while, etc.
- Le mot-clé **var** , quant à lui, permet de définir une variable globale ou locale à une fonction (sans distinction des blocs utilisés dans la fonction).
- Le mot-clé **const** sert à déclarer une référence constante.
 - Attention, une référence constante ne veut pas dire que la valeur derrière la référence est "immutable", mais que la référence elle-même est immutable.

La déclaration de variables globales, locales et constantes

Stocké en global ?

- **var** : oui ✓
- **let** : non ✗
- **const** : non ✗

Peut être réassigné ?

- **var** : oui ✓
- **let** : oui ✓
- **const** : non ✗

Se limite à la portée d'une fonction ?

- **var** : oui ✓
- **let** : oui ✓
- **const** : oui ✓

Se limite à la portée d'un block ?

- **var** : non ✗
- **let** : oui ✓
- **const** : oui ✓

Peut être redéclaré ?

- **var** : oui ✓
- **let** : non ✗
- **const** : non ✗

Redéclarer une variable est impossible sauf si c'est avec `var`, mais ce n'est pas une bonne pratique et il est conseillé d'éviter de le faire.

Exercice 7 – Création et Manipulation de variables globales, locales et constantes

```
//Exercice 7
/*Créer et manipuler des variables globales et locales a et b */
var a = 1;
var b = 2;
var c = 6;

if (a == 1) {
  const c=4; //constante qui ne peut pas être réassignée
  var a = 11; // la portée est globale
  let b = 22; // la portée est local au bloc de code if

  console.log(a); // 11
  console.log(b); // 22
  b = b+c;
  console.log(b); // 26
}

console.log(a); // 11
console.log(b); // 2
console.log(b+c); // 8
```

Attention:

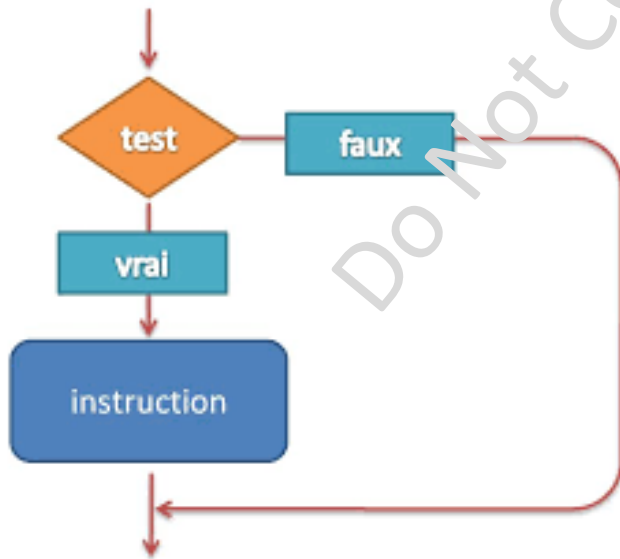
Erreur de syntax de re-declaration:

```
let x = 1;

{
  var x = 2;
}
```

Instructions conditionnelles If else

Une instruction conditionnelle (aussi appelé expression conditionnelle): est une fonction d'un langage de programmation, qui effectue différents calculs ou actions, en fonction de l'évaluation d'une condition booléenne, à savoir vraie ou fausse.



Exemple :

```
let x = 4;
let y = 0;

if((x > 1) == false){
  alert('x contient une valeur inférieure à 1');
}

if((x == y) == false){
  alert('x et y ne contiennent pas la même valeur');
}

if(y == false){
  alert('La valeur de y est évaluée à false');
}
```

Instructions conditionnelles If else

Structures conditionnelles simples

```
var x = 1;

if(x>1){
    alert('La valeur de x est supérieure à 1');
}else{
    alert('La valeur de x est inférieure à 1');
}
```

```
if(x>1){
    alert('La valeur de x est supérieure à 1');
}else if(x == 1){
    alert('La valeur de x est 1');
}else{
    alert('La valeur de x est inférieure à 1');
}
```

Structure conditionnelle multiple

```
switch(x){
    case 0 : alert('La valeur de x est supérieure à 1');
            break;
    case 1 : alert('La valeur de x est 1');
            break;
    case 2 : alert('La valeur de x est inférieure à 1');
            break;
    default : alert("Je ne sais pas !");
}
```

Exercice 8 – Instructions conditionnelles If else

```
//Exercice 8  
/* Ecrire un script qui permet à l'utilisateur de saisir son age  
et puis affiche dans le document "adulte" s'il est majeur ou mineur */
```

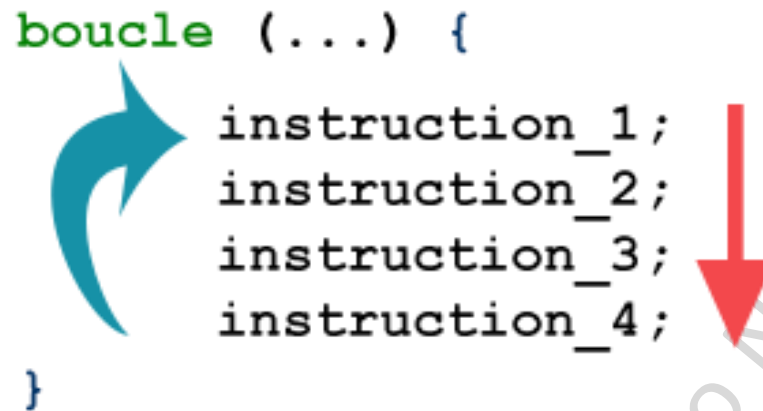
Exercice 8 – Instructions conditionnelles If else

```
//Exercice 8
/* Ecrire un script qui permet à l'utilisateur de saisir son age
et puis affiche dans le document "adulte" s'il est majeur ou mineur */
var age = prompt("Entrez votre âge :");
if(age>18){
    document.write("Vous êtes majeur");
    document.write('<br>');
}
else{
    document.write("Vous êtes mineur");
    document.writeln('<br>');
}
```

Boucle

Les boucles permettent de répéter des actions simplement et rapidement.
Il y a trois types de boucles:

- while
- for
- do while



```
boucle (...) {  
    instruction_1;  
    instruction_2;  
    instruction_3;  
    instruction_4;  
}
```

L'instruction **break** permet de terminer la boucle en cours.

Boucle

On peut créer une fonction grâce à « function »

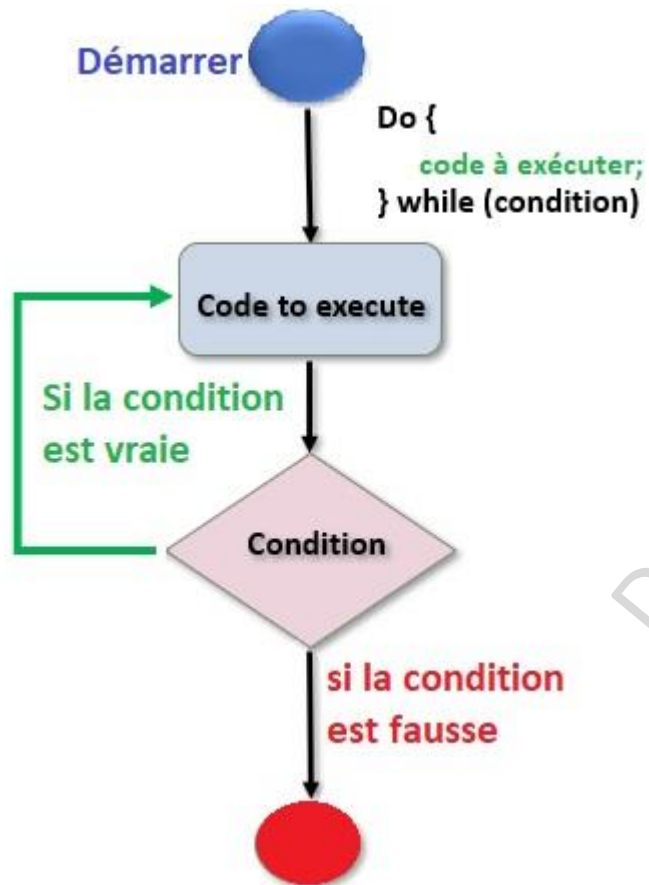
```
function aleatoire(){  
    return Math.random()*100;  
}  
  
function Multiplication(nombre1, nombre2){  
    return nombre1 + '*' + nombre2 + '=' + (nombre1 * nombre2);  
}  
  
document.getElementById('p1').innerHTML = aleatoire();  
document.getElementById('p2').innerHTML = Multiplication(5,10);
```

Une fonction peut être anonymes (ne pas avoir de nom correspondant)

```
var carré = function (n){  
    return n * n;  
}  
  
var x = carré(4);
```

Rq : La déclaration d'une fonction peut être faite après l'appel

Boucle do while



Exemple :

```
<!DOCTYPE html >
<html lang="fr">
<head>
<title>JavaScript</title>
<meta ="UTF-8">
</head>
<body>
<script>
  var i = 0;
  do {
    document.write(i + " ");
    i++;
  } while (i < 10);
</script>
</body>
</html>
```

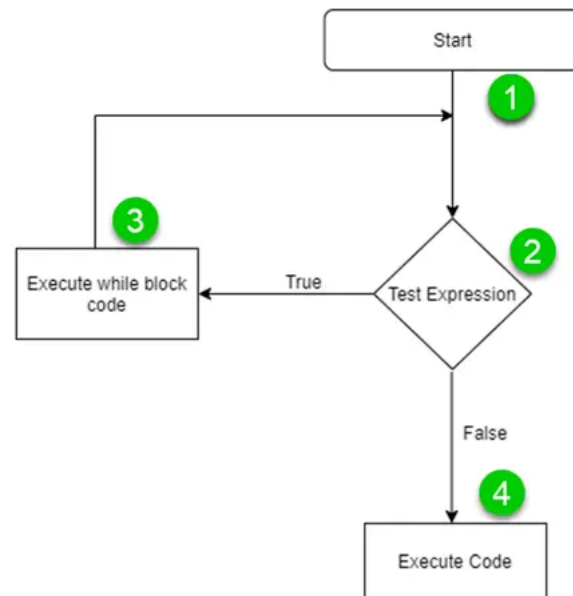
Boucle while

- La boucle do while est similaire à la boucle while, où un ensemble donné d'instructions est exécuté.
- La différence ici est que la boucle do while s'exécute complètement même si la condition n'est pas satisfaite.
- La boucle do while s'exécute jusqu'à ce que la condition spécifiée soit vraie et se termine dès que la condition n'est pas satisfaite.
- Pour terminer les tâches qui doivent être effectuées dans une itération, la boucle while peut être utilisée.

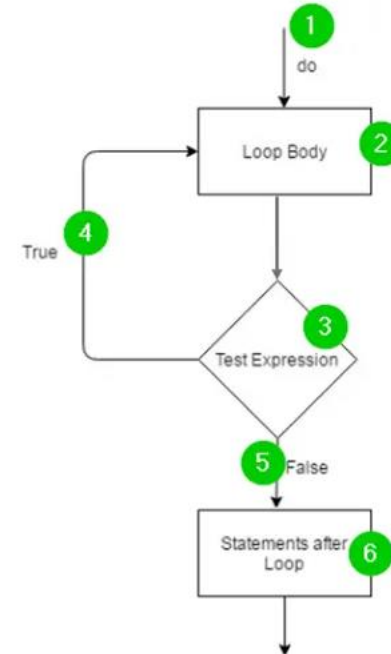
	While	Do While
Forme générale	<code>while (condition) {instructions;}</code>	<code>do {instructions;} while (Condition);</code>
Condition de contrôle	Dans la boucle 'while', la condition de contrôle est vérifié au début de la boucle.	Dans la boucle 'do while', la condition de contrôle est vérifié à la fin de la boucle.
Itérations	Les itérations ne se produisent pas si la condition à la première itération apparaît fausse.	L'itération se produit au moins une fois même si la condition est fausse à la première itération.

Boucle while

```
while (condition) {  
    statements;  
}
```



```
do {  
    statements  
} while (expression);
```



Boucle for

La condition qui suit FOR est composée de 3 éléments :

1. Une valeur ou expression initiale portant sur une variable entière appelée compteur.
2. Une condition : tant qu'elle est vraie, la répétition est poursuivie.
3. Une expression de poursuite qui consiste en la mise à jour du compteur.

```
for ([initialisation;][condition;][incrementation])  
{  
    instructions  
}
```

Exemple :

```
<!DOCTYPE html >  
<html lang="fr">  
<head>  
<title>JavaScript</title>  
<meta = "UTF-8">  
</head>  
<body>  
<script>  
    for (i=1; i<=50; i++) {  
        document.writeln(i+"<br>");  
    }  
</script>  
</body>  
</html>
```

Exercice 9 – Boucle For

```
//Exercice 9  
/*Utiliser for pour afficher tous les nombres entre 0 et 100 */
```

Exercice 9 – Boucle For

```
//Exercice 9
/*Utiliser for pour afficher tous les nombres entre 0 et 100 */
for (var i=0 ; i<=100 ; i++) {
    document.write(i);
    document.writeln('<br>');
}
```

Exercice 10 – Boucle For et Condition If

```
//Exercice 10
```

```
/*Utiliser for pour afficher les nombres pairs entre 0 et 100 */
```

Do Not Copy

Do Not Copy

Exercise 10 – Boucle For et Condition If

```
//Exercice 10
/*Utiliser for pour afficher les nombres pairs entre 0 et 100 */
for (var i=0 ; i<=100 ; i++) {
    if(i%2==0){
        document.write(i);
        document.writeln('<br>');
    }
}
```


Exercice 11 – Concaténation de string

```
//Exercice 11
/* Concaténation de string en utilisant l'opérateur + */
var prenom=prompt("Saisir votre prénom:");
var nom=prompt("Saisir votre nom:");
alert("Bonjour "+prenom+" "+nom+"!");
document.write("<br>Bonjour "+prenom+" "+nom+"!");
```

```
//Exercice 11
/* Concaténation de string en utilisant l'opérateur + */
var prenom=prompt("Saisir votre prénom:");
var nom=prompt("Saisir votre nom:");
alert("Bonjour "+prenom+" "+nom+"!");
/* document.write peut prendre plusieurs
paramètres d'entrée séparés par , */
document.write("<br>Bonjour ",prenom," ",nom,"!");
```

Fonction

- Les fonctions font partie des briques fondamentales de JavaScript.
- Une fonction est une procédure JavaScript, un ensemble d'instructions effectuant une tâche ou calculant une valeur.
- Afin d'utiliser une fonction, il est nécessaire de l'avoir auparavant définie au sein de la portée dans laquelle on souhaite l'appeler.

Fonction

Les déclarations de fonctions :

- Une définition de fonction (aussi appelée déclaration de fonction ou instruction de fonction) est construite avec le mot-clé **function**, suivi par :
 - Le nom de la fonction.
 - Une liste d'arguments à passer à la fonction, entre parenthèses et séparés par des virgules.
 - Les instructions JavaScript définissant la fonction, entre accolades, { }.

Fonction

- Le code suivant, par exemple, définit une fonction intitulée carré :

```
function carré(nombre) {  
    return nombre * nombre;  
}
```

L'instruction **return** spécifie la valeur qui est renvoyée par la fonction.

- **Appeler** la fonction permet d'effectuer les actions des instructions avec les paramètres indiqués.

```
var x = carré(4); //x reçoit la valeur 16
```

Exercice 12 – Fonction avec paramètre d'entrée

```
//Exercice 12
/*créer une fonction multipliePar2
qui prend comme paramètre d'entrée un valeur
et retourne son double.
Teste cette fonction. */
```

Exercice 12 – Fonction avec paramètre d'entrée

```
//Exercice 12
/*créer une fonction multipliePar2
qui prend comme paramètre d'entrée un valeur
et retourne son double.
Teste cette fonction. */

function multipliePar2(b){
    return b*2;
}
let x = 5;
document.write("le double de ",x," est ",multipliePar2(x));
```

Exercice 13 – Boucles While et Do While

```
//Exercice 13
/*Tester les boucles while et do while */
let resultat = '';
let i = 0;

do {
    i = i + 1;
    resultat = resultat + i;
} while (i>0 && i < 5);

console.log(resultat); // résultat attendu: "12345"

resultat='';
i = 0;
while (i>0 && i < 5) {
    i = i + 1;
    resultat = resultat + i;
}
```

Sarah Malaeb console.log(resultat); // résultat attendu: ""

Exercice 14 – Vérification de la saisie des données

• Solution 1

```
//Exercice 14
/*Vérifier la saisie de donnée */
function verif(saisie)
{
    if(saisie==' ' || isNaN(saisie))
        return false;
    else
        return true;
}

let n = prompt("Choisis un nombre:");
while(!verif(n))
{
    n = prompt("Recommence, il faut saisir un nombre non nul:");
}
alert('Le nombre saisi est '+n);
```


Exercice 14 – Vérification de la saisie des données

- **Solution 2**

```
//Exercice 14
/*Vérifier la saisie de donnée */
function verif(saisie)
{
    if(saisie=='')
    {
        saisie = verif(prompt("Recommence, il faut saisir un nombre non nul!"));
    }
    else if(isNaN(saisie))
    {
        saisie = verif(prompt("Recommence, ce n'est pas un nombre!"));
    }

    return saisie;
}

var nombre = verif(prompt("Choisis un nombre:"));
alert('Le nombre saisi est '+nombre);
```

Les événements détectables

- Les événements sont des actions ou des occurrences qui se produisent dans le système que vous programmez et dont le système vous informe afin que vous puissiez y répondre d'une manière ou d'une autre si vous le souhaitez.



- Par exemple, si l'utilisateur clique sur un bouton d'une page Web, vous pouvez répondre à cette action en affichant une boîte d'information.

Les événements détectables

- Petites fonctions déjà toutes faites.
- Exemple :

```
<body onload="alert('Bonjour')">
```

- Construction :

onLoad (au chargement)
= "ce qui doit se passer"

- onLoad s'exécute toujours dans le « BODY »

Les événements détectables

- **onChange**
 - Champs texte, zones texte, listes de sélection
 - S'exécute quand on change un élément de formulaire
- **onClick**
 - Boutons, boutons radio, boutons submit et reset, liens
 - S'exécute quand on clique dans ou sur un élément
- **onKeyDown**
 - Documents, images, liens, zones texte
 - S'exécute quand on appuie sur une touche du clavier
- **onKeyPress**
 - Documents, images, liens, zones texte
 - S'exécute quand on appuie et maintient une touche du clavier
- **onKeyUp**
 - Documents, images, liens, zones texte
 - S'exécute quand on relâche une touche du clavier

Les événements détectables

- **onLoad**
 - Documents
 - S'exécute quand le document se charge
- **onDragDrop**
 - Fenêtres
 - S'exécute quand on pose un élément à l'intérieur de la fenêtre du navigateur à l'aide la souris
- **onMove**
 - Fenêtres
 - S'exécute quand l'utilisateur ou un script bouge une fenêtre
- **onResize**
 - Fenêtres
 - S'exécute quand l'utilisateur ou un script change la taille d'une fenêtre
- **onUnload**
 - Documents
 - S'exécute quand on quitte le document

Les événements détectables

- **onMouseDown**
 - Documents, boutons, liens
 - S'exécute quand on clique avec le bouton de la souris
- **onMouseMove**
 - rien par défaut
 - S'exécute quand on bouge la souris
- **onMouseOut**
 - Cartes, liens
 - S'exécute quand le pointeur de la souris sort d'une zone de sélection graphique ou un lien
- **onMouseOver**
 - Liens
 - S'exécute quand le pointeur de la souris passe sur un lien
- **onMouseUp**
 - Documents, boutons, liens
 - S'exécute quand on relâche le bouton de la souris

Les événements détectables

- **onReset**
 - Formulaires
 - S'exécute quand on "resete" un formulaire
- **onSelect**
 - Champs ou zones texte
 - S'exécute quand on sélectionne une zone ou un champ texte (clavier ou souris)
- **onSubmit**
 - Formulaire
 - S'exécute au moment de l'envoi d'un formulaire

Les événements détectables

Exemple :

- Insérer une image pour un bouton
- Activer l'image quand le pointeur de la souris passe sur le lien et désactiver la quand il sort de la zone de sélection de cette image

```
<a href = "pageX.html "  
    onMouseOver="document.images['bouton1'].src='pic1.aktif.gif';"  
    onMouseOut="document.images['bouton1'].src='pic1.gif';">  
      
</a>
```


Les événements détectables

Autres exemples :

- alert: ouvre une fenêtre avec un message donné,
- modifier window.location pour aller vers une autre page,
- document.write: écrit un texte donné,
- setTimeout et setInterval: permettent respectivement de différer et de répéter une instruction ;

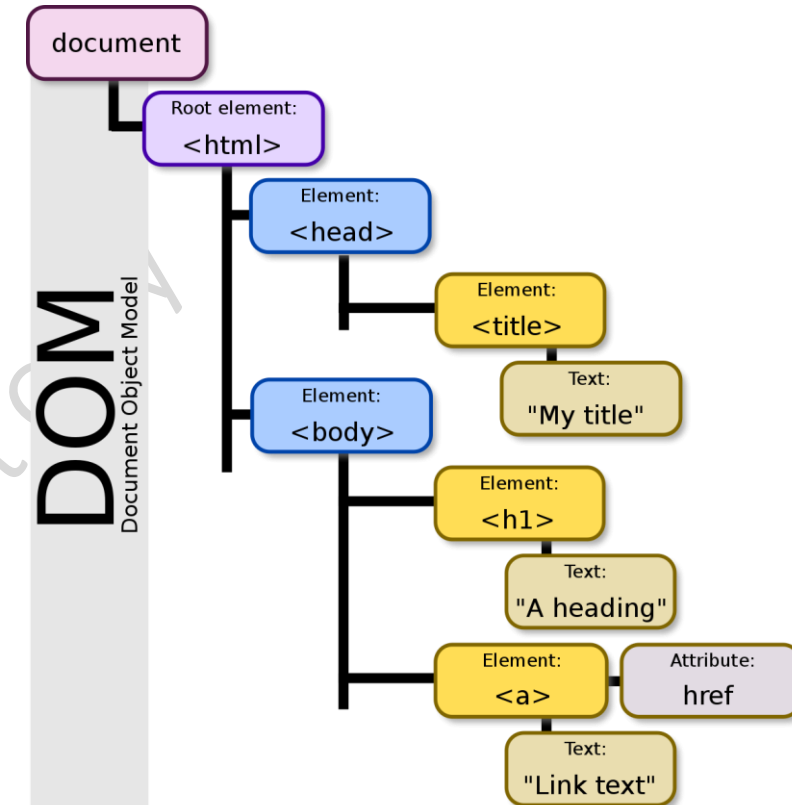
```
// déclenchement d'une alerte immédiate (alert)
<button onclick="alert('Coucou !')">cliquez ici !</button>

// déclenchement d'une alerte dans une demi-seconde (setTimeout)
<button onclick="setTimeout('alert(\'Surprise !\')',500)">
cliquez ici !</button>

// répétition toutes les 5 secondes (setInterval)
var clock_id = setInterval('alert(\'hi hi !\')',5000);
...
clearInterval(clock_id); // fin de la répétition
```

Rappel: Arborescence DOM

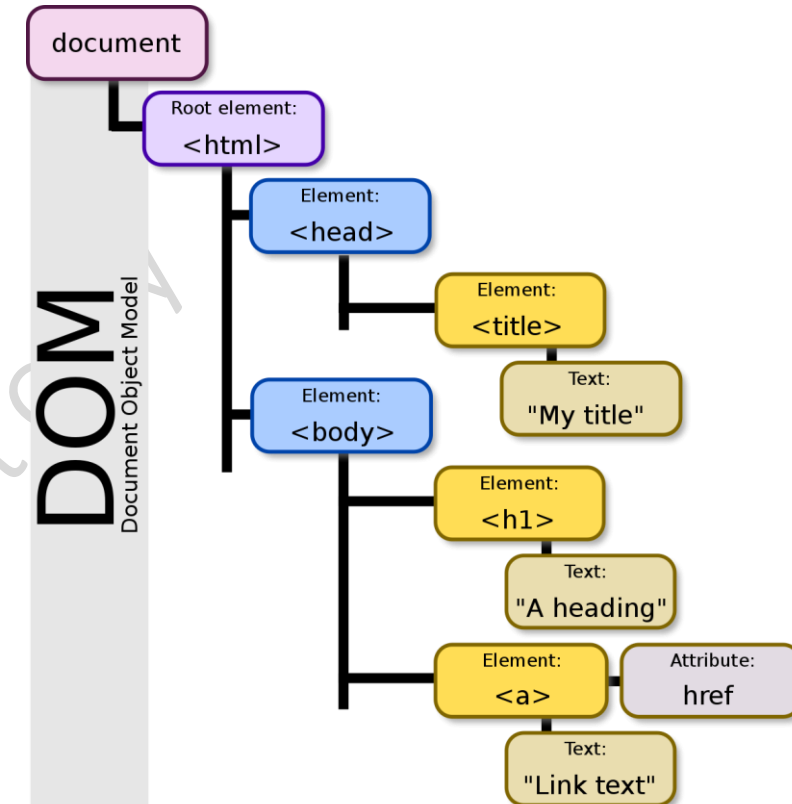
- Le modèle d'objet de document (Document Object Model)
- est une interface indépendante du langage qui traite un document HTML (ou XML) comme une **structure arborescente** dans laquelle **chaque nœud est un objet représentant une partie du document**.
- Le DOM représente un document avec une arborescence logique.



Accès à un élément dans le DOM

- par son attribut id
- par son attribut class
- par sa balise
- par un sélecteur CSS
- par son attribut name

Fonction	Description
getElementById(id)	Renvoie un élément par son id
getElementsByTagName(balise)	Renvoie la liste des éléments du document dont la balise est « balise »
createTextNode(texte)	Crée une balise texte
createElement(balise)	Crée une balise HTML
getElementsByClassName(className)	Renvoie la liste des éléments par nom de classe



Travail en autonomie

- Continuer l'activité en laboratoire informatique

Do Not Copy

Do Not Copy