



RÉSUMÉ THÉORIQUE – FILIÈRE DÉVELOPPEMENT DIGITAL

M105 – Programmer en Javascript

 120 heures





SOMMAIRE



01 - DÉFINIR LE RÔLE DE JAVASCRIPT DANS LE DÉVELOPPEMENT

Comparer un langage de script avec un langage compilé
Comprendre l'architecture client/serveur
Découvrir l'écosystème de développement

02 - ACQUÉRIR LES FONDAMENTAUX DE JAVASCRIPT

Maîtriser la syntaxe JavaScript et ses notions fondamentales
Maîtriser les structures de contrôle
Utiliser des fonctions & Manipuler les objets

03 - MANIPULER LES ÉLÉMENTS D'UNE PAGE AVEC DOM

Comprendre l'arbre DOM, les nœuds parents et enfants
Connaître les bases de la manipulation du DOM en JavaScript
Manipuler les éléments HTML

04 - GÉRER LES ÉVÉNEMENTS UTILISATEUR

Comprendre la notion d'événement pour gérer l'interactivité
Gérer les éléments d'un formulaire

05 - MANIPULER JQUERY

Découvrir jQuery
Découvrir AJAX

MODALITÉS PÉDAGOGIQUES



1



2



3



4



5

Le guide de soutien

Il s'agit du résumé théorique et du manuel des travaux pratiques.

La version PDF

Une version PDF du guide de soutien est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life.

Des ressources téléchargeables

Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life.

Du contenu interactif

Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life.

Des ressources en lignes

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage.



PARTIE 1

DÉFINIR LE RÔLE DE JAVASCRIPT DANS LE DÉVELOPPEMENT

Dans ce module, vous allez :

- Comparer un langage de script avec un langage compilé
- Comprendre l'architecture client/serveur
- Découvrir l'écosystème de développement



12 heures

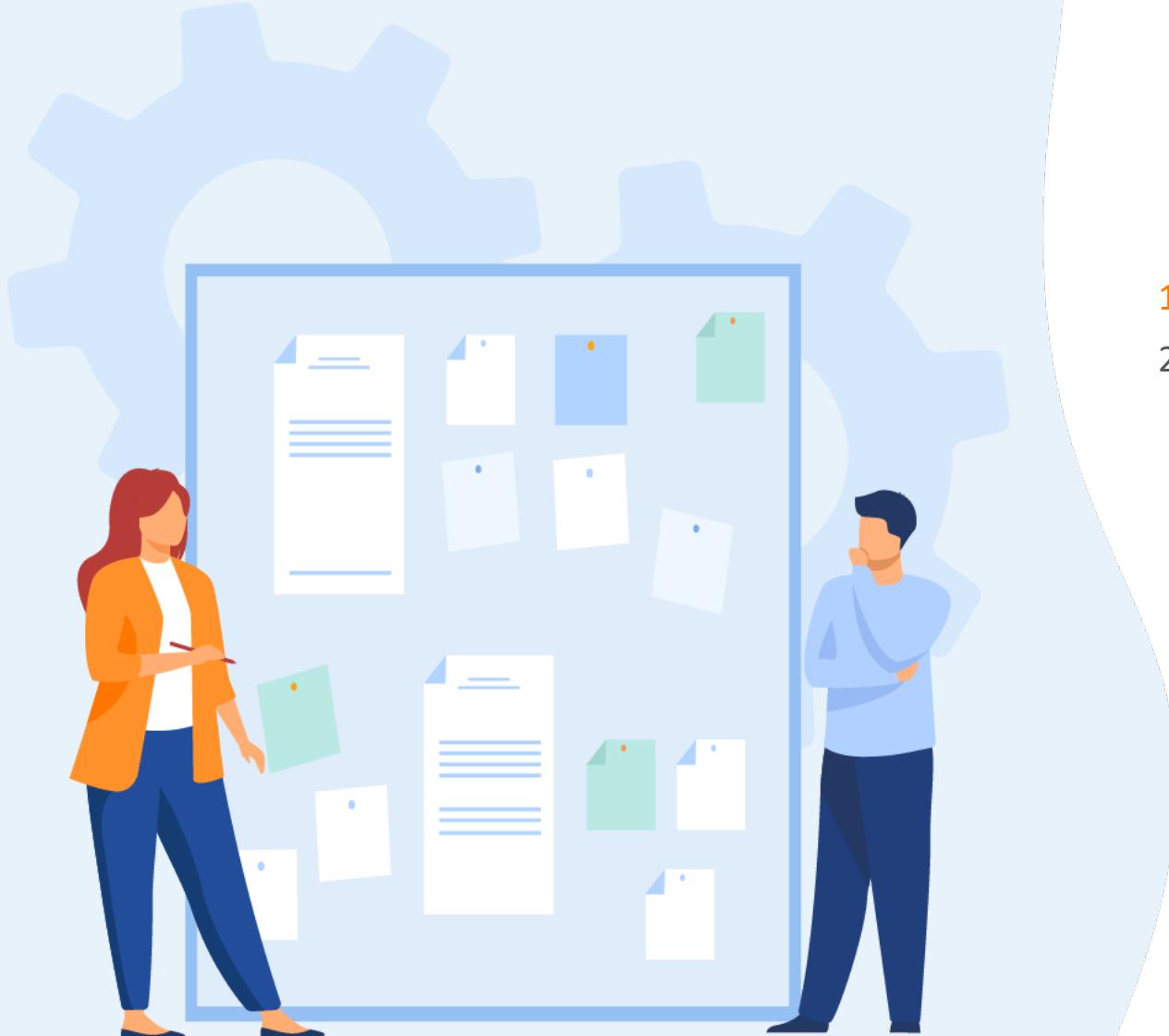


CHAPITRE 1

COMPARER UN LANGAGE DE SCRIPT AVEC UN LANGAGE COMPILE

Ce que vous allez apprendre dans ce chapitre :

- Définir un langage de script
- Fonctionnement d'un langage de script



CHAPITRE 1

COMPARER UN LANGAGE DE SCRIPT AVEC UN LANGAGE COMPILE

1. Définir un langage de script
2. Fonctionnement d'un langage de script

01 – Langage de script vs langage compilé

Définir un langage de script



tous les langages de script sont des langages de programmation. La différence théorique entre les deux réside dans le fait que les langages de script ne nécessitent pas l'étape de compilation et sont plutôt interprétés.

Par exemple, un programme C doit être compilé avant d'être exécuté, alors qu'un langage de script tel que JavaScript ou PHP n'a pas besoin d'être compilé.

Généralement, les programmes compilés s'exécutent plus rapidement que les programmes interprétés, car ils sont d'abord convertis en code machine natif. D'ailleurs, les compilateurs lisent et analysent le code une seule fois et signalent les erreurs que le code pourrait avoir, mais l'interpréteur lira et analysera les instructions de code chaque fois qu'il les rencontrera et s'arrêtera à cette instance même s'il y a une erreur.

Exemple des langages de script traditionnellement utilisés sans étape de compilation explicite sont PHP, JavaScript, Python, VBScript.

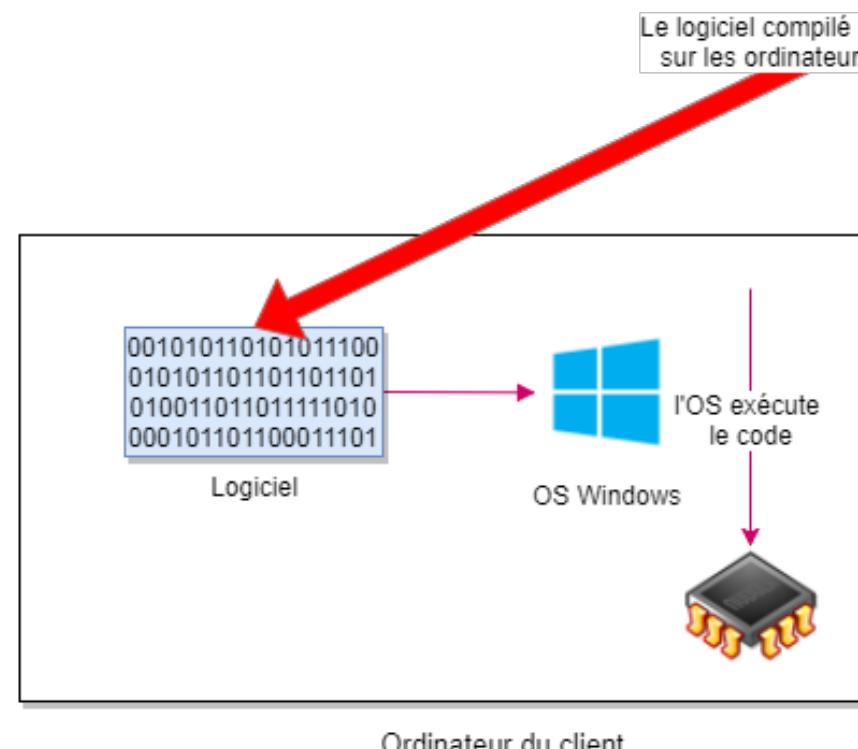
Différence entre compilateur et interpréteur

- **Compilateurs** : convertit d'abord l'ensemble du programme en code d'assembleur, puis convertit le code d'assembleur en code machine.
- **Interpréteur** : convertit et exécute le programme ligne par ligne.

01 – Langage de script vs langage compilé

Définir un langage de script

Langage compilé

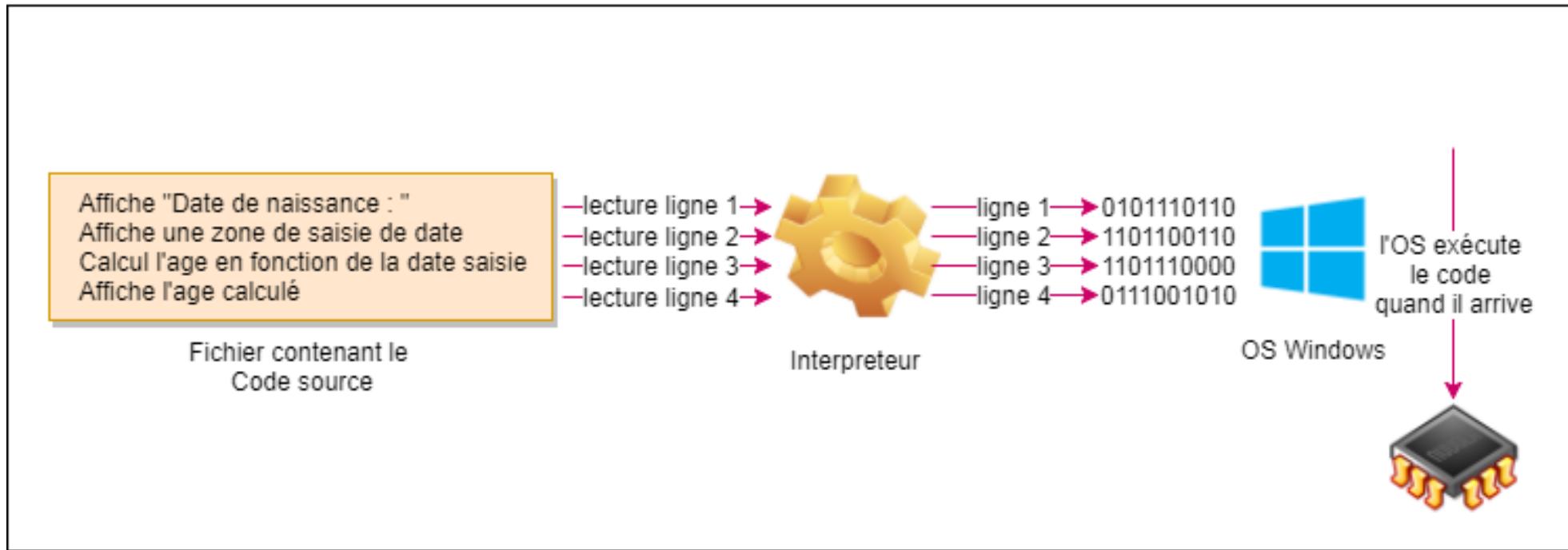


01 – Langage de script vs langage compilé

Définir un langage de script

Langage interprété

Ordinateur





CHAPITRE 1

COMPARER UN LANGAGE DE SCRIPT AVEC UN LANGAGE COMPILE

1. Définir un langage de script
2. Fonctionnement d'un langage de script

01 – Langage de script vs langage compilé

Fonctionnement d'un langage de script



L'interpréteur

Un interpréteur se distingue d'un compilateur par le fait qu'il effectue l'analyse et la traduction nécessaires à l'exécution d'un programme donné non pas une fois pour toutes, mais à chaque exécution de ce programme. L'exécution nécessite ainsi de disposer non seulement du programme, mais aussi de l'interprète correspondant.

Le cycle d'un interprète est le suivant :

- lire et analyser une instruction (ou expression) ;
- si l'instruction est syntaxiquement correcte, l'exécuter (ou évaluer l'expression) ;
- passer à l'instruction suivante.

L'intérêt des langages interprétés réside principalement dans la facilité de programmation et dans la portabilité. Les langages interprétés facilitent énormément la mise au point des programmes car ils évitent la phase de compilation, souvent longue, et limitent les possibilités de bogues. Il est en général possible d'exécuter des programmes incomplets, ce qui facilite le développement rapide d'applications ou de prototypes d'applications.



CHAPITRE 2

COMPRENDRE L'ARCHITECTURE CLIENT/SERVEUR

Ce que vous allez apprendre dans ce chapitre :

- Composition d'une architecture client/serveur
- Fonctionnement d'un système client/serveur pour le cas d'une architecture Web



CHAPITRE 2

COMPRENDRE L'ARCHITECTURE CLIENT/SERVEUR

1. Composition d'une architecture client/serveur
2. Fonctionnement d'un système client/serveur pour le cas d'une architecture Web

02 – COMPRENDRE L'ARCHITECTURE

CLIENT/SERVEUR

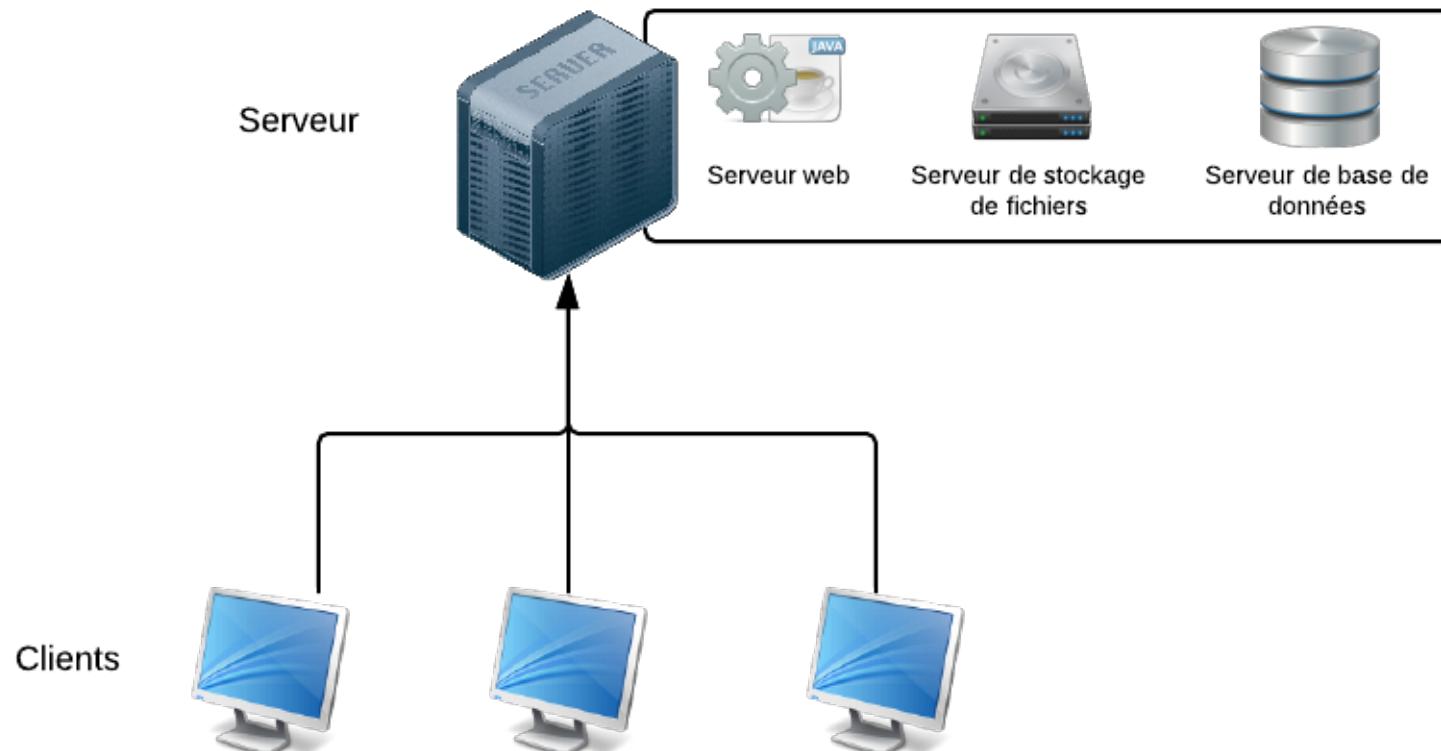
Composition d'une architecture client/serveur

Définition de l'architecture Client/Serveur

Concrètement, il s'agit de plusieurs machines reliées par un réseau local ou étendu. Certaines de ces machines sont capables d'offrir des services, tels que l'exécution d'une procédure, la recherche de données, le partage d'une ressource : ce sont des serveurs.

D'autres vont utiliser ces services : ce sont les clients.

Si l'on s'en tient à la définition toute machine peut être client ou serveur ou les deux à la fois ou alternativement l'un puis l'autre.





CHAPITRE 2

IDENTIFIER LES APPROCHES D'ANALYSE D'UN PROBLÈME

1. Composition d'une architecture client/serveur
2. Fonctionnement d'un système client/serveur pour le cas d'une architecture Web

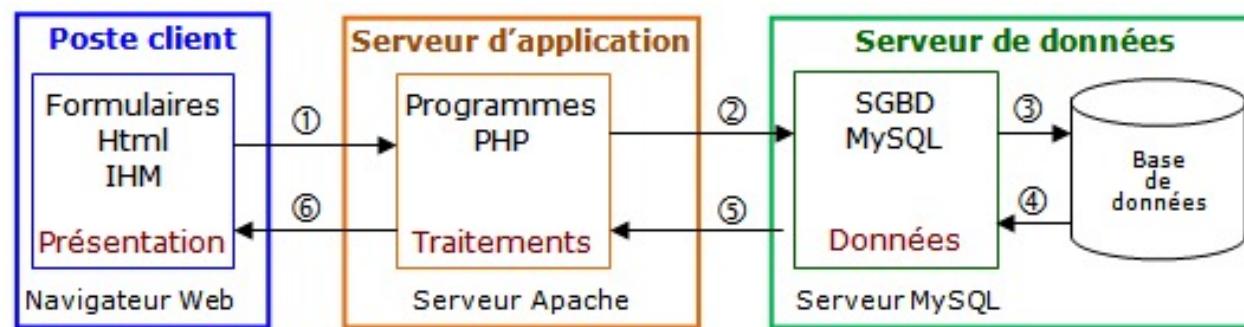
02 – COMPRENDRE L'ARCHITECTURE

CLIENT/SERVEUR

Fonctionnement d'un système client/serveur Web

Fonctionnement

- Un système client/serveur fonctionne selon le schéma suivant:
- Le client émet une requête vers le serveur grâce à son adresse et à son port, qui désigne un service particulier du serveur
- Le serveur reçoit la demande et répond à l'aide de l'adresse de la machine client (et de son port)



- ① requête http demandée par l'utilisateur au travers de l'activation d'une URL
- ② requête SQL
- ③ accès aux données
- ④ données résultantes
- ⑤ jeux d'enregistrement contenant les données brutes
- ⑥ document html contenant les données qui seront visualisées dans le navigateur Web

02 – COMPRENDRE L'ARCHITECTURE

CLIENT/SERVEUR

Fonctionnement d'un système client/serveur Web



Serveurs Web et HTTP

Les navigateurs web communiquent avec les serveurs web avec le protocole HTTP : HyperTextTransfer Protocol. Quand vous cliquez un lien sur une page, soumettez un formulaire ou lancez une recherche, le navigateur envoie une requête HTTP (HTTP Request) au serveur.

Cette requête inclut :

- Une URL identifiant la cible et la ressource (un fichier HTML, un point particulier de données sur le serveur ou un outil à lancer).
- Une méthode qui définit l'action requise (par exemple récupérer un fichier ou sauvegarder certaines données ou mises à jour). Les différentes méthodes/verbes et les actions associées sont listées ci-dessous :
 - GET: Récupérer une ressource spécifique, par exemple un fichier html contenant des informations sur un produit ou une liste de produits.
 - POST: Crée une ressource comme un nouvel article dans un wiki, ajouter un contact dans une base de données, enregistrer les données d'un formulaire d'inscription...
 - HEAD: Récupérer les informations "metadata" d'une ressource spécifique sans le "body" comme ferait GET. Vous pouvez utiliser une requête HEAD pour, par exemple, la date de dernière mise à jour d'une ressource puis, utiliser GET (plus "coûteuse") seulement si la ressource a été changée.
 - PUT: Met à jour une ressource existante ou en crée une si elle n'existe pas.
 - DELETE: Supprime la ressource spécifiée.

02 – COMPRENDRE L'ARCHITECTURE

CLIENT/SERVEUR

Fonctionnement d'un système client/serveur Web



Serveurs Web et HTTP

Les serveurs Web attendent une requête du client puis la traitent quand elle arrive. Il répond ensuite au navigateur avec un message HTTP Response. La réponse contient un statut HTTP Response indiquant si, oui ou non, la requête a abouti. (ex : "200 OK" pour un succès, "404 Not Found" si la ressource ne peut être trouvée, "403 Forbidden" si l'utilisateur n'est pas autorisé à voir la ressource etc. Le corps d'une réponse aboutie à une requête GET contiendrait la ressource demandée.

Quand une page HTML est retournée, elle est affichée par le navigateur. Le navigateur, nativement, pourra découvrir des liens vers d'autres ressources (ex : une page HTML intègre habituellement des pages JavaScript et CSS), et enverra des requêtes séparées pour télécharger ces fichiers.

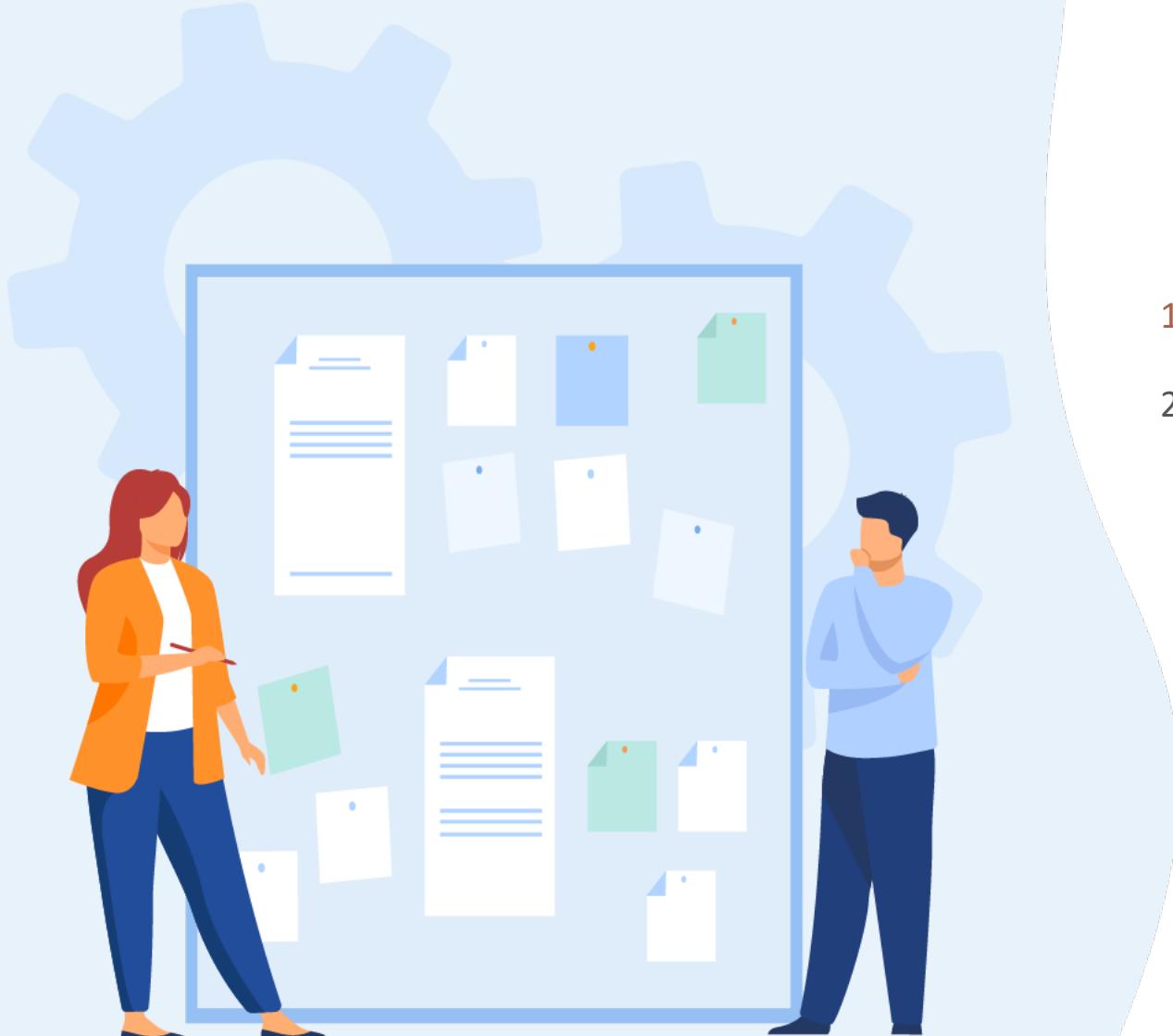


CHAPITRE 3

DÉCOUVRIR L'ÉCOSYSTÈME DE DÉVELOPPEMENT

Ce que vous allez apprendre dans ce chapitre :

- Environnements de développement
- Découverte des librairies appropriés (jQuery, React, Vue JS, Angular, ...)



CHAPITRE 2

DÉCOUVRIR L'ÉCOSYSTÈME DE DÉVELOPPEMENT

1. Environnements de développement
2. Découverte des librairies appropriés (jQuery, React, Vue JS, Angular, ...)

Choix de l'environnement de développement

Techniquement on peut faire de la programmation web sur les 3 systèmes d'exploitations les plus utilisées aujourd'hui à savoir Windows, Linux et macOS. Mais je vous recommande tout de même de choisir plutôt Linux ou macOS.

Pourquoi?

Parce que la plupart des sites Internet et des applications web sont hébergés et tournent sur des environnements Linux.

Et il est fort à parier que ce sera votre cas. Pour minimiser les problèmes de compatibilités au niveau des déploiements, il vaut mieux travailler sur un environnement local équivalent de la production.

Mac OS étant extrêmement proche de Linux, ses 2 OS sont donc à privilégier.

DÉCOUVRIR L'ÉCOSYSTÈME DE DÉVELOPPEMENT

Environnements de développement

Les langages de programmation web sur lesquels se focaliser

Il existe beaucoup de langages de programmation pour faire du développement web, mais il faut bien comprendre une distinction.

Le web fonctionne en 3 étapes :

1. Le client effectue une demande au serveur
2. Le serveur traite la demande du client et lui renvoie une réponse
3. Le client reçoit la réponse et la traite pour l'afficher

Le serveur représente la partie Backend et le client la partie Frontend. Il y a donc des langages de programmation orientée front et orientée back.





CHAPITRE 2

DÉCOUVRIR L'ÉCOSYSTÈME DE DÉVELOPPEMENT

1. Environnements de développement
2. Découverte des librairies appropriés (jQuery, React, Vue JS, Angular, ...)

02 – COMPRENDRE L'ARCHITECTURE

CLIENT/SERVEUR

Découverte des librairies appropriés



Les frameworks backend

React : React est un langage frontend qui fonctionne au sein de JS pour améliorer les fonctionnalités d'utilisation. Il s'agit d'une bibliothèque open-source qui a été initialement publiée en 2013. Il a été développé par Facebook il y a environ 8 ans et est maintenant utilisé pour les principales applications web telles que Facebook, Instagram, WhatsApp, Yahoo ! Etc.

Angular: Angular est un outil moderne de développement frontend qui est de plus en plus populaire dans de nombreuses applications avec la disposition « feed ». Il devient de plus en plus populaire après sa sortie initiale en 2016.

De grands noms comme Microsoft et Autodesk font actuellement appel à cette technologie. Angular vous aide à créer des applications dynamiques à page unique (SPAS) et interactives grâce à ses caractéristiques étonnantes.

Vue: Vue est l'un des principaux langages frontend dynamiques pour la création d'une interface spécifique. Pour la liaison de données active, Vue a été initialement publié en 2014. De plus, ce framework pourrait être exploité sur un constructeur d'électrons. Il prend également en charge les applications de bureau et les applications mobiles.

Vue s'inspire d'Angular et de React, qui sont tous deux très bien établis sur le marché. Cela signifie qu'il peut soutenir et permettre une compréhension plus facile pour les nouveaux développeurs qui cherchent à mettre en œuvre cette technologie.

Jquery: jQuery est une petite bibliothèque qui peut être intégrée dans un fichier JavaScript. Cela modifie le comportement de JavaScript et sa fonctionnalité. En effet, cet outil peut également être utilisé pour des projets de programmation de bout en bout.



PARTIE 2

ACQUÉRIR LES FONDAMENTAUX DE JAVASCRIPT

Dans ce module, vous allez :

- Maîtriser la syntaxe JavaScript et ses notions fondamentales
- Maîtriser les structures de contrôle
- Utiliser des fonctions
- Manipuler les objets



 48 heures



CHAPITRE 1

MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Ce que vous allez apprendre dans ce chapitre :

- Notions de variables et de données
- Expressions et opérateurs
- Notions de lecture et d'écriture
- Types primitifs et objets de base

12 heures



CHAPITRE 1

MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

1. Notions de variables et de données
2. Expressions et opérateurs
3. Notions de lecture et d'écriture
4. Types primitifs et objets de base

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de variables et de données



Ecrivez un programme JavaScript pour afficher le jour et l'heure dans le format suivant. Allez dans l'éditeur

Exemple de sortie : Aujourd'hui est : Mardi.

L'heure actuelle est : 22h:30:38

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de variables et de données



Identifiants JavaScript

Toutes les variables JavaScript doivent être identifiées avec des noms uniques .

Ces noms uniques sont appelés identifiants .

Les identifiants peuvent être des noms courts (comme x et y) ou des noms plus descriptifs (age, sum, totalVolume).

Les règles générales de construction des noms de variables (identifiants uniques) sont :

- Les noms peuvent contenir des lettres, des chiffres, des traits de soulignement et des signes dollar.
- Les noms doivent commencer par une lettre
- Les noms peuvent aussi commencer par \$ et _ (mais nous ne l'utiliserons pas dans ce tutoriel)
- Les noms sont sensibles à la casse (y et Y sont des variables différentes)
- Les mots réservés (comme les mots-clés JavaScript) ne peuvent pas être utilisés comme noms

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de variables et de données



Types de données JavaScript

Les variables JavaScript peuvent contenir des nombres comme 100 et des valeurs de texte comme "Hassan ALAMI".

En programmation, les valeurs de texte sont appelées chaînes de texte.

JavaScript peut gérer de nombreux types de données, mais pour l'instant, pensez simplement aux nombres et aux chaînes.

Les chaînes sont écrites entre guillemets simples ou doubles. Les nombres sont écrits sans guillemets.

Si vous mettez un nombre entre guillemets, il sera traité comme une chaîne de texte.

```
var pi = 3.14;
var person = "Hassan ALAMI";
var answer = 'C\'est moi!';
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de variables et de données



Valeur = **undefined**

Dans les programmes informatiques, les variables sont souvent déclarées sans valeur. La valeur peut être quelque chose qui doit être calculé, ou quelque chose qui sera fourni plus tard, comme une entrée utilisateur.

Une variable déclarée sans valeur aura la valeur **undefined**.

La variable carName aura la valeur undefined après l'exécution de cette instruction :

```
var ville;
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de variables et de données



La déclaration avec LET

Avant ES6 (2015), JavaScript n'avait que Global Scope et Function Scope .

ES6 a introduit deux nouveaux mots clés JavaScript importants : let et const.

Ces deux mots-clés fournissent Block Scope en JavaScript.

Les variables déclarées à l'intérieur d'un bloc { } ne sont pas accessibles depuis l'extérieur du bloc :

```
{  
  let x = 2;  
}  
// x n'est pas accessible ici
```

Les variables déclarées avec le mot - clé var NE PEUVENT PAS avoir de portée de bloc.

Les variables déclarées à l'intérieur d'un bloc { } sont accessibles depuis l'extérieur du bloc.

```
{  
  var x = 2;  
}  
// x est accessible ici
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de variables et de données



La déclaration avec CONST

Le mot-clé **const** a été introduit dans ES6 (2015) .

Les variables définies avec const ne peuvent pas être redéclarées.

Les variables définies avec const ne peuvent pas être réaffectées.

Les variables définies avec const ont une portée de bloc.:

```
const PI = 3.141592653589793;  
PI = 3.14;      // Erreur  
PI = PI + 10;  // Erreur
```

Les variables JavaScript const doivent se voir attribuer une valeur lorsqu'elles sont déclarées :

```
const PI = 3.14159265359; // Correct
```

```
const PI ;  
PI= 3.14159265359; // Incorrect
```

La déclaration d'une variable avec const est similaire à let. Les deux ont une portée de bloc.



CHAPITRE 1

MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

1. Notions de variables et de données
2. Expressions et opérateurs
3. Notions de lecture et d'écriture
4. Types primitifs et objets de base

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Expressions et opérateurs



Opérateurs arithmétiques JavaScript

Les opérateurs d'affectation affectent des valeurs aux variables JavaScript.

Operateur	Description
+	Addition
-	Soustraction
*	Multiplication
**	Puissance (ES2016)
/	Division
%	Modulo (Reste de la division)
++	Incrémentation
--	Décrémentations

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Expressions et opérateurs



Opérateurs d'affectation JavaScript

Les opérateurs arithmétiques sont utilisés pour effectuer des opérations arithmétiques sur des nombres :

Operateur	Exemple
=	x = y
+=	x += y
-=	x -= y
*=	x *= y
/=	x /= y
%=	x %= y
**=	x **= y
=	x = y

L' opérateur d' affectation d'addition (+=) ajoute une valeur à une variable.

```
let x = 10; //x=10  
x += 5; //x=15
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Expressions et opérateurs



Opérateurs de chaîne JavaScript

L'opérateur + peut également être utilisé pour ajouter (concaténer) des chaînes.

```
let text1 = "CASA";
let text2 = "BLANCA";
let text3 = text1 + " " + text2;
```

L'opérateur += d'affectation peut également être utilisé pour ajouter (concaténer) des chaînes :

```
let text1 = "CASA ";
text1 += "BLANCA";
```

L'ajout de deux nombres renvoie la somme, mais l'ajout d'un nombre et d'une chaîne renvoie une chaîne :

```
let x = 5 + 5; //x=10
let y = "5" + 5; //y="55"
let z = "Hello" + 5; //z="Hello5"
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Expressions et opérateurs



Opérateurs de comparaison JavaScript

L'opérateur + peut également être utilisé pour ajouter (concaténer) des chaînes.

Operateur	Exemple
==	Etal à (valeur)
====	Egal à (valeur et type)
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

```
let x = 5;
let y = "5";
let z=(x==y); //z=true
let z=(x====y); //z=false
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Expressions et opérateurs



Opérateurs logiques JavaScript

L'opérateur + peut également être utilisé pour ajouter (concaténer) des chaînes.

Operateur	Exemple
&&	ET logique
	OU logique
!	NON logique

Opérateurs de type JavaScript

Operateur	Exemple
typeof	Retourner le type de la variable
instanceof	Retourner true si l'objet est une instance de classe donnée en paramètre

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Expressions et opérateurs

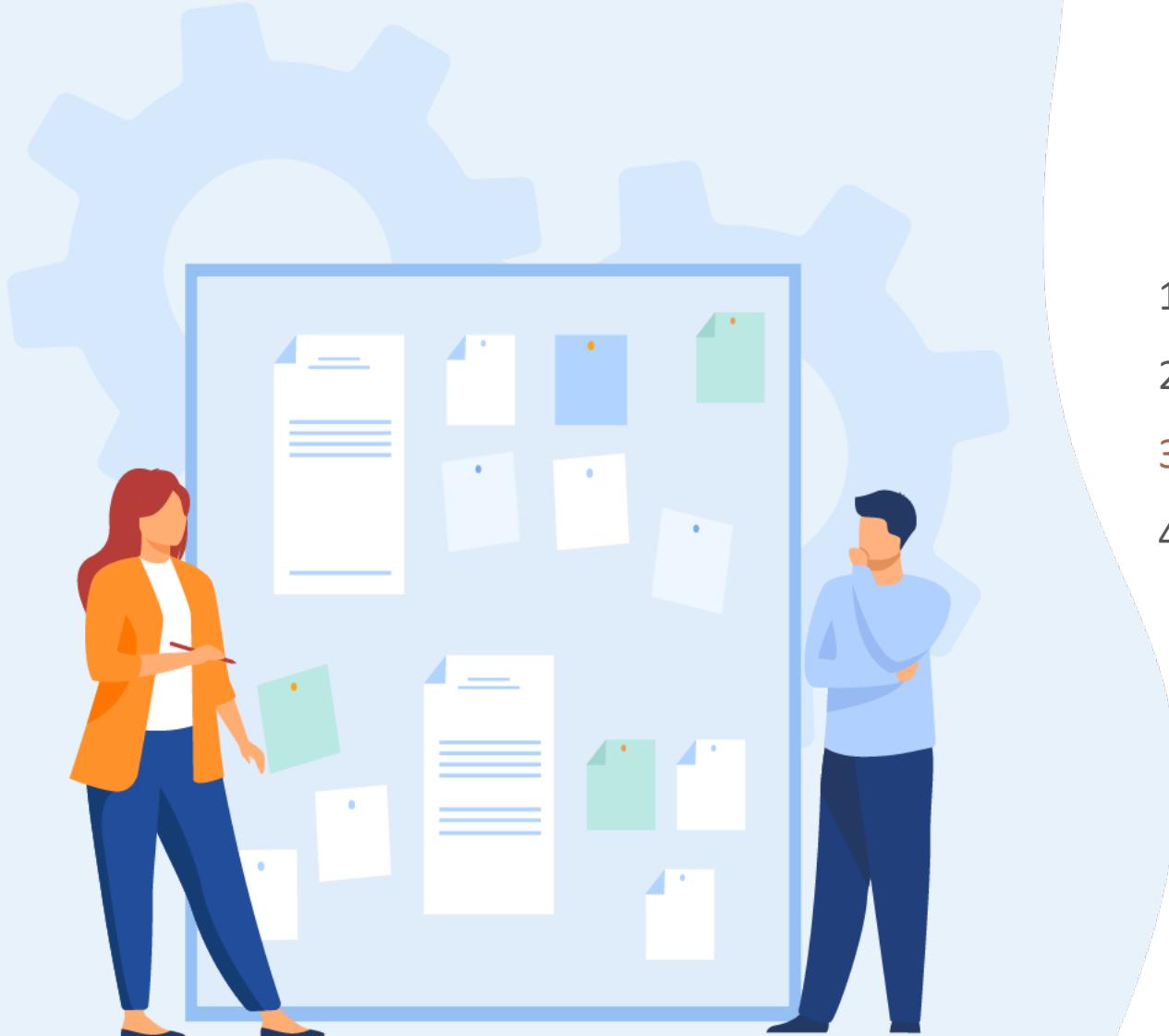


Opérateurs au niveau du bit JavaScript

Les opérateurs de bits fonctionnent sur des nombres de 32 bits.

Tout opérande numérique de l'opération est converti en un nombre de 32 bits. Le résultat est reconvertis en un nombre JavaScript.

Operateur	Description	Exemple	Comme	Résultat	Décimal
&	ET binaire	5 & 1	0101 & 0001	0001	1
	OU binaire	5 1	0101 0001	0101	5
~	NON binaire	~5	~0101	1010	10
^	XOR binaire	5 ^ 1	0101 ^ 0001	0100	4
<<	Décalage de bits à gauche	5 << 1	0101 << 1	1010	10
>>	Décalage de bits à droite	5 >> 1	0101 >> 1	0010	2



CHAPITRE 1

MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

1. Notions de variables et de données
2. Expressions et opérateurs
3. Notions de lecture et d'écriture
4. Types primitifs et objets de base

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de lecture et d'écriture



Possibilités d'affichage JavaScript

JavaScript peut « afficher » les données de différentes manières :

- Écrire dans un élément HTML, à l'aide de **innerHTML**.
- Écriture dans la sortie HTML à l'aide de **document.write()**.
- Écriture dans une boîte d'alerte, à l'aide de **window.alert()**.
- Écriture dans la console du navigateur, à l'aide de **console.log()**.

Utiliser innerHTML

Pour accéder à un élément HTML, JavaScript peut utiliser la méthode **document.getElementById(id)**.

L'attribut id définit l'élément HTML. La propriété **innerHTML** définit le contenu HTML :

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de lecture et d'écriture



Utilisation de document.write()

À des fins de test, il est pratique d'utiliser document.write():

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
document.write(5 + 6);
</script>

</body>
</html>
```

L'utilisation de document.write() après le chargement d'un document HTML supprimera tout le HTML existant :

Utilisation de window.alert()

Vous pouvez utiliser une boîte d'alerte pour afficher des données :

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

Utilisation de console.log()

À des fins de débogage, vous pouvez appeler la méthode console.log()

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de lecture et d'écriture



Impression JavaScript

JavaScript n'a pas d'objet d'impression ni de méthode d'impression.

Vous ne pouvez pas accéder aux périphériques de sortie à partir de JavaScript.

La seule exception est que vous pouvez appeler la méthode **window.print()** dans le navigateur pour imprimer le contenu de la fenêtre en cours.

```
<!DOCTYPE html>
<html>
<body>

<button onclick="window.print()">Imprimer cette page</button>

</body>
</html>
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de lecture et d'écriture



Les entrées Javascript

Javascript récupère les données à partie de deux méthodes: prompt et <input>

Prompt

La méthode prompt() affiche une boîte de dialogue qui invite le visiteur à entrer.

Une boîte de dialogue est souvent utilisée si vous souhaitez que l'utilisateur saisisse une valeur avant de saisir une page.

Remarque : Lorsqu'une boîte de dialogue s'affiche, l'utilisateur doit cliquer sur « OK » ou « Annuler » pour continuer après avoir entré une valeur d'entrée. N'abusez pas de cette méthode, car elle empêche l'utilisateur d'accéder à d'autres parties de la page jusqu'à ce que la boîte soit fermée.

La méthode prompt() renvoie la valeur d'entrée si l'utilisateur clique sur "OK". Si l'utilisateur clique sur « annuler », la méthode renvoie null.

```
var prenom = prompt("Quel est votre prénom?");  
document.write(prenom);
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de lecture et d'écriture



Les entrées Javascript

Javascript récupère les données à partie de deux méthodes: prompt et <input>

<input>

Javascript peut récupérer les données de n'importe quel élément html par la méthode: **document.getElementById(id)**.
Il faut juste spécifier l'identifiant "id" dans le code HTML

```
<!DOCTYPE html>
<html>
<body>
<input type="text" id="prenom">
<button onclick="alert(document.getElementById('prenom').value)">Afficher</button>
</body>
</html>
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Notions de lecture et d'écriture



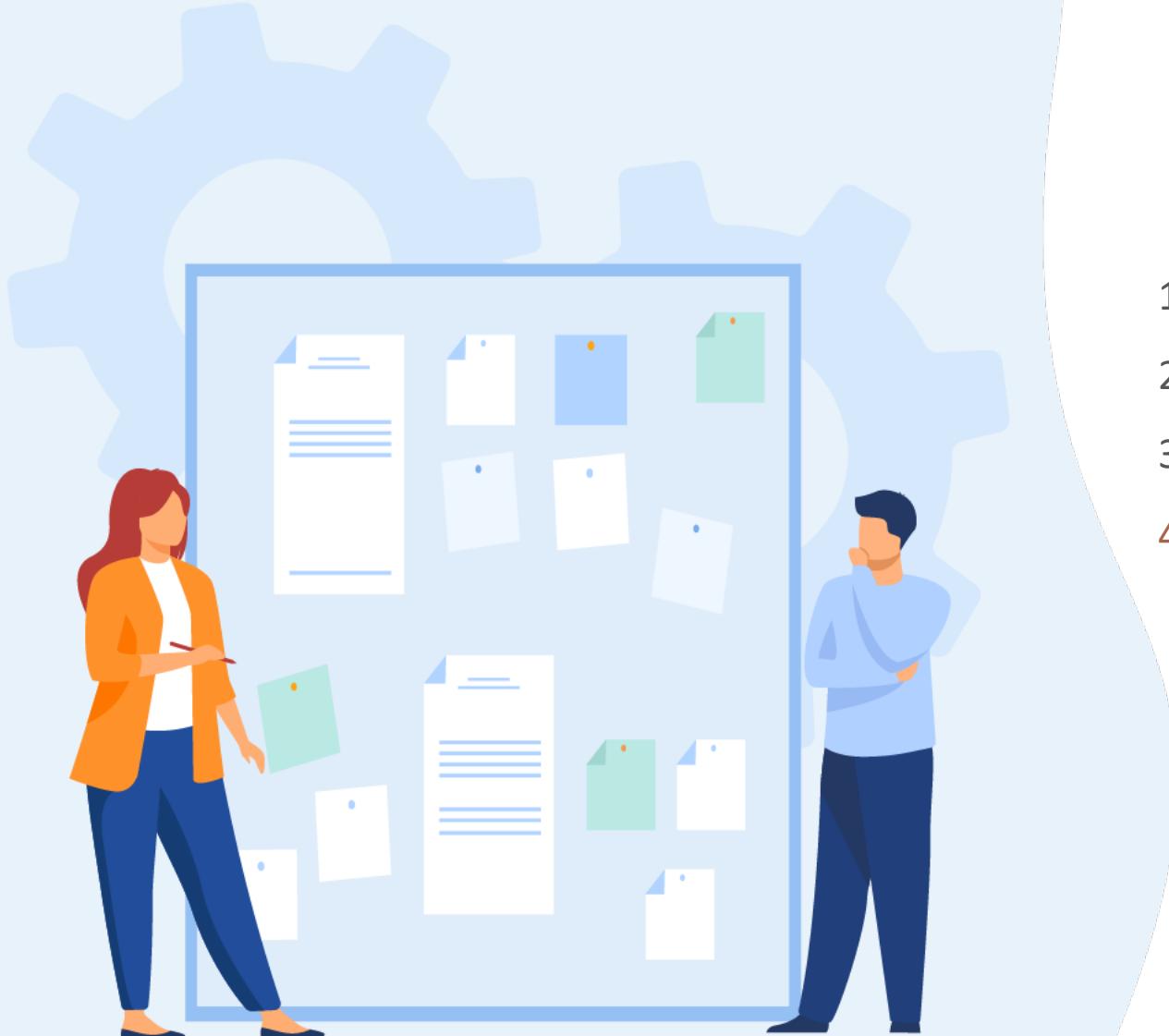
Récupérer le contenu d'une balise <p>

innerHTML est le contenu d'une balise

```
<html>
<body>
<p id="paragraphe">
Maroc
</p>
<button onclick="alert(document.getElementById('paragraphe').innerHTML)">Afficher</button>
</body>
</html>
```

Vous pouvez également récupérer la valeur de toutes les propriétés telles que l'alignement, les dimensions le styles, ...

```
<html>
<body>
<p id="paragraphe" align="center" width="100px" style="color:red">
Maroc
</p>
<button onclick="alert(document.getElementById('paragraphe').style.color)">Afficher la couleur</button>
<button onclick="alert(document.getElementById('paragraphe').align)">Afficher l'alignement</button>
<button onclick="alert(document.getElementById('paragraphe').width)">Afficher la largeur</button>
</body>
</html>
```



CHAPITRE 1

MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

1. Notions de variables et de données
2. Expressions et opérateurs
3. Notions de lecture et d'écriture
4. Types primitifs et objets de base

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



En JavaScript, il existe 5 types de données différents pouvant contenir des valeurs :

- string
- number
- boolean
- object
- function

Il existe 6 types d'objets :

Object

- Date
- Array
- String
- Number
- Boolean

Et 2 types de données qui ne peuvent pas contenir de valeurs :

- null
- undefined

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



Données primitives

Une valeur de données primitive est une valeur de données simple et unique sans propriétés ni méthodes supplémentaires.

L'opérateur **typeof** peut renvoyer l'un de ces types primitifs :

- string
- number
- boolean
- undefined

```
typeof "Hassan"           // Retourne "string"
typeof 3.14                // Retourne "number"
typeof true                 // Retourne "boolean"
typeof false                // Retourne "boolean"
typeof x                    // Retourne "undefined" (if x has no value)
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



Données complexes

L'opérateur `typeof` peut renvoyer l'un des deux types complexes :

- `function`
- `object`

L'opérateur `typeof` renvoie "object" pour les objets, les tableaux et null.

L'opérateur `typeof` ne renvoie pas "object" pour les fonctions.

```
typeof {name:'Hassan', age:34} // Retourne "object"
typeof [1,2,3,4]             // Retourne "object" (not "array", see note below)
typeof null                   // Retourne "object"
typeof function myFunc(){}    // Retourne "function"
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



Données complexes

L' opérateur typeof peut renvoyer l'un des deux types complexes :

- function
- object

L' opérateur typeof renvoie "object" pour les objets, les tableaux et null.

L' opérateur typeof ne renvoie pas "object" pour les fonctions.

```
typeof {name:'Hassan', age:34} // Retourne "object"
typeof [1,2,3,4]           // Retourne "object" (not "array", see note below)
typeof null                 // Retourne "object"
typeof function myFunc(){}  // Retourne "function"
```

Différence entre non défini et nul

Undefined et null sont de valeur égale mais de type différent :

```
typeof undefined          // undefined
typeof null               // object
null === undefined        // false
null == undefined         // true
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



Conversion de type JavaScript

Les variables JavaScript peuvent être converties en une nouvelle variable et un autre type de données :

Par l'utilisation d'une fonction JavaScript

Automatiquement par JavaScript lui-même

Conversion de chaînes en nombres

La méthode globale Number() peut convertir des chaînes en nombres.

Les chaînes contenant des nombres (comme "3.14") sont converties en nombres (comme 3.14).

Les chaînes vides sont converties en 0.

Tout le reste est converti en NaN (pas un nombre).

```
Number("3.14")    // retourne 3.14
Number(" ")       // retourne 0
Number("")        // retourne 0
Number("99 88")   // retourne NaN
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



L'opérateur unaire +

L'opérateur unaire + peut être utilisé pour convertir une variable en nombre :

```
let y = "5";           // y is a string
let x = + y;          // x is a number
```

Si la variable ne peut pas être convertie, elle deviendra quand même un nombre, mais avec la valeur NaN (Pas un nombre) :

Conversion de nombres en chaînes

La méthode globale String() peut convertir des nombres en chaînes.

Il peut être utilisé sur tout type de nombres, de littéraux, de variables ou d'expressions :

```
String(x)           // retourne une string
String(123)          // retourne une string
String(100 + 23)    // retourne une string
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



L'opérateur unaire +

L'opérateur unaire + peut être utilisé pour convertir une variable en nombre :

```
let y = "5";           // y est une string
let x = + y;          // x est un number
```

Si la variable ne peut pas être convertie, elle deviendra quand même un nombre, mais avec la valeur NaN (Pas un nombre) :

Conversion de nombres en chaînes

La méthode globale String() peut convertir des nombres en chaînes.

Il peut être utilisé sur tout type de nombres, de littéraux, de variables ou d'expressions :

```
String(x)           // retourne une string
String(123)          // retourne une string
String(100 + 23)    // retourne une string
```

La méthode Number toString() fait la même chose.

```
x.toString()
(123).toString()
(100 + 23).toString()
```

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



Conversion de dates en nombres

La méthode globale Number() peut être utilisée pour convertir des dates en nombres.

```
d = new Date();
Number(d)          // returns 1404568027739
```

La méthode de date **getTime()** fait de même.

Conversion de dates en chaînes

La méthode globale String() peut convertir des dates en chaînes.

```
String(Date()) // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)"
```

La méthode Date **toString()** fait de même.

01 – MAÎTRISER LA SYNTAXE JAVASCRIPT ET SES NOTIONS FONDAMENTALES

Types primitifs et objets de base



Conversion de dates en chaînes

Method	Description
getDate()	Renvoie le jour comme nombre(1-31)
getDay()	Renvoie le numéro de jour de la semaine comme nombre(0-6)
getFullYear()	Renvoie l'année en 4 chiffres(yyyy)
getHours()	Renvoie l'heure (0-23)
getMilliseconds()	Renvoie les milisecondes (0-999)
getMinutes()	Renvoie les minutes (0-59)
getMonth()	Renvoie le nombre (0-11)
getSeconds()	Renvoie les secondes(0-59)
getTime()	Renvoie les secondes depuis 1970



CHAPITRE 2

MAÎTRISER LES STRUCTURES DE CONTRÔLE

Ce que vous allez apprendre dans ce chapitre :

- Structures alternatives
- Structures itératives



12 heures



CHAPITRE 2

MAÎTRISER LES STRUCTURES DE CONTRÔLE

1. Structures alternatives
2. Structures itératives

02 – MAÎTRISER LES STRUCTURES DE CONTRÔLE

Structures alternatives



La déclaration if

Utilisez l'instruction if pour spécifier un bloc de code JavaScript à exécuter si une condition est vraie.

Syntaxe

```
if (condition) {  
    // bloc d'instructions à executer si la condition est vraie  
}
```

Exemple

On affiche "Réussi" si la note est supérieure ou égal à 10 :

```
if (note >= 10) {  
    document.write("Réussi");  
}
```

02 – MAÎTRISER LES STRUCTURES DE CONTRÔLE

Structures alternatives



La déclaration else

Utilisez l'instruction else pour spécifier un bloc de code à exécuter si la condition est fausse.

Syntaxe

```
if (condition) {  
    // bloc d'instructions à executer si la condition est vraie  
} else {  
    // bloc d'instructions à executer si la condition est fausse  
}
```

Exemple

On affiche "Réussi" si la note est supérieure ou égal à 10, sinon on affiche "Echoué"

```
if (note >= 10) {  
    document.write("Réussi");  
} else {  
    document.write("Echoué");  
}
```

02 – MAÎTRISER LES STRUCTURES DE CONTRÔLE

Structures alternatives



L'instruction else if

Utilisez l'instruction else if pour spécifier une nouvelle condition si la première condition est fausse.

Syntaxe

```
if (condition1) {  
    // bloc d'instructions à executer si la condition1 est vraie  
} else if (condition2) {  
    // bloc d'instructions à executer si la condition1 est vraie et la condition2 est fausse  
} else {  
    // bloc d'instructions à executer si la condition1 est vraie et la condition2 est fausse  
}
```

Exemple

```
if (note >= 10) {  
    document.write("Réussi")  
} else if (note>8) {  
    document.write("rattrapage")  
}  
else{  
    document.write("Echoué")  
}
```

02 – MAÎTRISER LES STRUCTURES DE CONTRÔLE

Structures alternatives



L'instruction switch

Utilisez l'instruction switch pour sélectionner l'un des nombreux blocs de code à exécuter.

Syntaxe

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

Voilà comment cela fonctionne:

L'expression de commutateur est évaluée une fois.

La valeur de l'expression est comparée aux valeurs de chaque cas.

S'il y a correspondance, le bloc de code associé est exécuté.

S'il n'y a pas de correspondance, le bloc de code par défaut est exécuté.

Lorsque JavaScript atteint un break mot - clé, il sort du bloc de commutation.

Exemple

```
switch (new Date().getDay()) {  
    case 0:  
        day = "Sunday"; break;  
    case 1:  
        day = "Monday"; break;  
    case 2:  
        day = "Tuesday"; break;  
    case 3:  
        day = "Wednesday"; break;  
    case 4:  
        day = "Thursday"; break;  
    case 5:  
        day = "Friday"; break;  
    case 6:  
        day = "Saturday";  
}
```



CHAPITRE 2

MAÎTRISER LES STRUCTURES DE CONTRÔLE

1. Structures alternatives
2. Structures itératives

02 – MAÎTRISER LES STRUCTURES DE CONTRÔLE

Structures itératives



Différents types de boucles

JavaScript prend en charge différents types de boucles :

for - parcourt un bloc de code plusieurs fois

for/in - parcourt les propriétés d'un objet

for/of - parcourt les valeurs d'un objet itérable

while - parcourt un bloc de code tant qu'une condition spécifiée est vraie

do/while - parcourt également un bloc de code tant qu'une condition spécifiée est vraie

02 – MAÎTRISER LES STRUCTURES DE CONTRÔLE

Structures itératives



La boucle For

Exemple

```
for (let i = 0; i < 5; i++) {  
    text += "Le nombre est " + i + "<br>";  
}
```

Array.forEach()

Exemple

```
const numbers = [45, 4, 9, 16, 25];  
let txt = "";  
numbers.forEach(myFunction);  
function myFunction(value, index, array) {  
    txt += value;  
}
```

Array.forEach()

Exemple

```
const langages = ["Java", "Python", "C++"];  
let text = "";  
for (let x of langages) {  
    text += x;  
}
```

La boucle For in

```
const person = {fname:"Hassan", lname:"FILALI", age:25};  
  
let text = "";  
for (let x in person) {  
    text += person[x];  
}
```

Exemple expliqué

La boucle for in itère sur un objet person
Chaque itération renvoie une clé (x)
La clé est utilisée pour accéder à la valeur de la clé
La valeur de la clé est person[x]

```
const numbers = [45, 4, 9, 16, 25];  
  
let txt = "";  
for (let x in numbers) {  
    txt += numbers[x];  
}
```

02 – MAÎTRISER LES STRUCTURES DE CONTRÔLE

Structures itératives



La déclaration Break

L'instruction **break** peut également être utilisée pour sortir d'une boucle :

Exemple

```
for (let i = 0; i < 10; i++) {  
    if (i === 3) { break; }  
    text += "The number is " + i + "<br>";  
}
```

Dans l'exemple ci-dessus, l'instruction `break` termine la boucle ("interrompt" la boucle) lorsque le compteur de boucle (`i`) est 3.

La déclaration continue

L'instruction `continue` interrompt une itération (dans la boucle), si une condition spécifiée se produit, et continue avec l'itération suivante dans la boucle.

Cet exemple ignore la valeur 3 :

Exemple

```
for (let i = 0; i < 10; i++) {  
    if (i === 3) { continue; }  
    text += "The number is " + i + "<br>";  
}
```



CHAPITRE 3

UTILISER DES FONCTIONS

Ce que vous allez apprendre dans ce chapitre :

- Fonctions
- Expressions lambdas
- Appels asynchrones (callBack, Promise)
- Gestion des exceptions



12 heures



CHAPITRE 3

UTILISER DES FONCTIONS

1. Fonctions
2. Expressions lambdas
3. Appels asynchrones (callBack, Promise)
4. Gestion des exceptions

03 – UTILISER DES FONCTIONS

Fonctions

Définition

- Une fonction est un ensemble de code, qui est défini une fois et peut être appelé n nombre de fois. Une fonction peut être réutilisée, ainsi elles sont également utilisées pour éviter la répétition du code.
- Les fonctions sont l'un des principaux piliers de JavaScript. Fondamentalement, une fonction JavaScript est un ensemble d'instructions qui effectue certaines tâches ou effectuent des calculs, puis renvoie le résultat à l'utilisateur
- Dans JavaScript , une fonction est sensiblement la même qu'une procédure ou un sous-programme, dans d'autres langages de programmation.
- La fonction JavaScript est un ensemble d'instructions utilisées pour effectuer une tâche spécifique. Il peut prendre une ou plusieurs entrées et peut également renvoyer une sortie . Les deux, prendre une entrée et renvoyer une sortie sont facultatifs.
- Une fonction JavaScript est exécutée lorsqu'elle est appelée.

03 – UTILISER DES FONCTIONS

Fonctions

Définition

- Une fonction JavaScript est un bloc de code conçu pour effectuer une tâche particulière.
- Une fonction JavaScript est exécutée lorsque "quelque chose" l'invoque (l'appelle).

Syntaxe des fonctions JavaScript

Une fonction JavaScript est définie avec le mot - clé **function**, suivi d'un nom , suivi de parenthèses () .

Les noms de fonction peuvent contenir des lettres, des chiffres, des traits de soulignement et des signes dollar (mêmes règles que les variables).

Les parenthèses peuvent inclure des noms de paramètres séparés par des virgules :
(paramètre1, paramètre2, ...)

Le code à exécuter, par la fonction, est placé entre accolades : {}

```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

Les paramètres de fonction sont répertoriés entre parenthèses () dans la définition de la fonction.

Les arguments de fonction sont les valeurs reçues par la fonction lorsqu'elle est invoquée.

A l'intérieur de la fonction, les arguments (les paramètres) se comportent comme des variables locales.

03 – UTILISER DES FONCTIONS

Fonctions



Appel de fonction

- Le code à l'intérieur de la fonction s'exécutera lorsque « quelque chose » invoque (appelle) la fonction :
- Lorsqu'un événement se produit (quand un utilisateur clique sur un bouton)
- Lorsqu'il est invoqué (appelé) à partir du code JavaScript
- Automatiquement (auto-invoqué)

Retour de la fonction

Lorsque JavaScript atteint une instruction **return**, la fonction arrête de s'exécuter.

Si la fonction a été invoquée à partir d'une instruction, JavaScript "retournera" pour exécuter le code après l'instruction d'appel.

Les fonctions calculent souvent une valeur de retour . La valeur de retour est "renvoyée" à "l'appelant":

```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

Exemple

```
let x = myFunction(4, 3); // La fonction est appelée, la valeur renournée est affectée à x  
function myFunction(a, b) {  
    return a * b;           // La fonction retourne le produit de a et b  
}
```

03 – UTILISER DES FONCTIONS

Fonctions



Appel de fonction

- Le code à l'intérieur de la fonction s'exécutera lorsque « quelque chose » invoque (appelle) la fonction :
- Lorsqu'un événement se produit (quand un utilisateur clique sur un bouton)
- Lorsqu'il est invoqué (appelé) à partir du code JavaScript
- Automatiquement (auto-invoqué)

Retour de la fonction

Lorsque JavaScript atteint une instruction **return**, la fonction arrête de s'exécuter.

Si la fonction a été invoquée à partir d'une instruction, JavaScript "retournera" pour exécuter le code après l'instruction d'appel.

Les fonctions calculent souvent une valeur de retour . La valeur de retour est "renvoyée" à "l'appelant":

```
function name(parameter1, parameter2, parameter3) {  
    // code to be executed  
}
```

Exemple

```
let x = myFunction(4, 3); // La fonction est appelée, la valeur renournée est affectée à x  
function myFunction(a, b) {  
    return a * b;           // La fonction retourne le produit de a et b  
}
```

03 – UTILISER DES FONCTIONS

Fonctions



Variables locales

- Les variables déclarées dans une fonction JavaScript deviennent LOCALES à la fonction.
- Les variables locales ne sont accessibles qu'à partir de la fonction.

```
// Le code ici ne peut pas utiliser la variable "nom"

function myFunction() {
    let nom = "Hassan";
    // Le code ici peut utiliser la variable "nom"
}

// Le code ici ne peut pas utiliser la variable "nom"
```

Étant donné que les variables locales ne sont reconnues qu'à l'intérieur de leurs fonctions, les variables portant le même nom peuvent être utilisées dans différentes fonctions.

Les variables locales sont créées lorsqu'une fonction démarre et supprimées lorsque la fonction est terminée.



CHAPITRE 3

UTILISER DES FONCTIONS

1. Fonctions
2. Expressions lambdas
3. Appels asynchrones (callBack, Promise)
4. Gestion des exceptions

03 – UTILISER DES FONCTIONS

Expressions lambdas

- Une fonction fléchée est une alternative compacte à une expression de fonction traditionnelle, mais elle est limitée et ne peut pas être utilisée dans toutes les situations.
- Au lieu de continuer à utiliser le mot fonction, nous pouvons l'omettre, mais à la place, nous devons mettre un signe égal (=) plus une parenthèse carrée fermante (>) [ou un “plus grand que” plus connu] après la parenthèse fermante:

Exemple1

```
const variable = () => {
  return "ma_variable"
}

console.log(variable()) // "ma_variable"
```

Exemple2

```
const variable = (a,b) => {
  return a*b;
}

console.log(variable(2,3)) // 6
```

Exemple3

```
const langages = [
  'Java',
  'Python',
  'PHP',
  'Scala'
];

console.log(langages.map(L => L.length));
```

La fonction map parcourt les éléments de la liste "langages" et applique l'expression lambda défini par la fonction: (L)=>L.length

03 – UTILISER DES FONCTIONS

Expressions lambdas

Exemple4

```
const langages = [
  'Java',
  'Python',
  'PHP',
  'Scala'
];
langages.forEach(L=>{
  if (L.length>4){
    console.log(L);
  }
})
```



CHAPITRE 3

UTILISER DES FONCTIONS

1. Fonctions
2. Expressions lambdas
3. Appels asynchrones (callBack, Promise)
4. Gestion des exceptions

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)

Fonction de rappel (Callback)

Une fonction de rappel (aussi appelée callback en anglais) est une fonction passée dans une autre fonction en tant qu'argument, qui est ensuite invoquée à l'intérieur de la fonction externe pour accomplir une sorte de routine ou d'action.

Voici un rapide exemple :

```
const array = [1,2,3,4,5];

function change(tab,callback){
    const arr=[];
    for(i=0;i<tab.length;i++){
        arr.push(callback(tab[i]));
    }
    return arr;
}

const resultat = change(array,e=>e*2);
console.log(resultat);
```

La fonction change se concentre sur le parcours du tableau et laisse la fonction callback faire le traitement.
On peut utiliser plusieurs fonctions callback en paramètres

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)



Fonction de rappel (Callback)

Dans la pratique, les callbacks sont le plus souvent utilisés avec des fonctions asynchrones.

Un exemple typique est JavaScript setTimeout().

```
setTimeout(myFunction, 3000);

function myFunction() {
    document.getElementById("demo").innerHTML = "Good job";
}
```

Dans l'exemple ci-dessus, myFunction est utilisé comme rappel.

La fonction (le nom de la fonction) est passée setTimeout() en argument.

3000 est le nombre de millisecondes avant le délai d'attente, il myFunction() sera donc appelé après 3 secondes.

Lorsque vous passez une fonction en argument, n'oubliez pas de ne pas utiliser de parenthèses.

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)



Fonction de rappel (Callback)

Exemple avec setInterval

```
setInterval(myFunction, 1000);

function myFunction() {
    let d = new Date();
    document.getElementById("demo").innerHTML=
        d.getHours() + ":" +
        d.getMinutes() + ":" +
        d.getSeconds();
}
```

Dans l'exemple ci-dessus, myFunction est utilisé comme rappel.

La fonction (le nom de la fonction) est passée setInterval() en argument.

1000 est le nombre de millisecondes entre les intervalles, il myFunction() sera donc appelé toutes les secondes.

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)



Les promesses

En JavaScript, une promesse est un objet qui renvoie une valeur que vous espérez recevoir dans le futur, mais pas maintenant.

Étant donné que la valeur sera renvoyée par la promesse à l'avenir, la promesse est très bien adaptée à la gestion des opérations asynchrones.

Il sera plus facile de comprendre le concept des promesses JavaScript par un exemple concret.

Soit le traitement suivant.

Instruction1

Instruction2

Instruction3 //Qui récupère une valeur externe à la
mettre dans un élément HTML (cela prend 5 secondes)

Instruction4

La situation actuelle provoque une attente de 5 seconds pour que l'instruction4 soit exécutée.

L'objectif des promesses est de laisser l'instruction3 s'exécuter et en même temps continuer l'exécution du programme. Quand l'instruction3 récupère la valeur depuis la ressource externe, elle sera placée dans l'élément de l'interface

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)

Les promesses

Une promesse a trois états :

- **En attente** : vous ne savez pas si vous terminerez votre traitement de l'instruction3
- **Accompli** : vous terminez avec succès l'instruction3.
- **Rejeté** : vous avez arrêté l'instruction3.

Une promesse commence dans l'état en attente, ce qui indique que la promesse n'est pas terminée. Il se termine par un état rempli (réussi) ou rejeté (échec).

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)



Créer une promesse : le constructeur Promise

Pour créer une promesse en JavaScript, vous utilisez le constructeur Promise :

```
let completed = true;

let getData = new Promise(function (resolve, reject) {
    if (completed) {
        resolve("Donnée récupérée");
    } else {
        reject("Je n'ai pas pu récupérer la donnée");
    }
});
```

Le constructeur Promise accepte une fonction comme argument. Cette fonction s'appelle le **executor**.

L'exécuteur accepte deux fonctions avec les noms, par convention, resolve() et reject().

Lorsque vous appelez le **new Promise(executor)**, le **executor** est automatiquement appelé.

À l'intérieur de l'exécuteur, vousappelez manuellement la fonction resolve() si l'exécuteur est terminé avec succès etappelez la fonction reject() en cas d'erreur.

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)



Créer une promesse : le constructeur Promise

Si vous intégrez le code JavaScript ci-dessus dans un document HTML et vérifiez la fenêtre de la console, vous verrez que la promesse est résolue car la variable completed est définie sur true.

Pour voir l'état d'attente de la promesse, nous encapsulons le code de l'exécuteur dans la fonction `setTimeout()` :

```
let completed = true;

let getData = new Promise(function (resolve, reject) {
    setTimeout(() => {
        if (completed) {
            resolve("Donnée récupérée");
        } else {
            reject("Je n'ai pas pu récupérer la donnée");
        }
    }, 3000);
});
```

Maintenant, vous voyez que la promesse commence par l'état pending avec la valeur est undefined. La valeur promise sera renvoyée ultérieurement une fois la promesse terminée.

```
▼ Promise {<pending>} ⓘ
  ► __proto__: Promise
    [[PromiseStatus]]: "pending"
    [[PromiseValue]]: undefined
```

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)



Créer une promesse : le constructeur Promise

Après environ 3 secondes, tapez le **getData** dans la fenêtre de la console, vous verrez que l'état de la promesse devient resolved et la valeur de la promesse est la chaîne que nous avons passée à la fonction resolve().

```
▼ Promise {<pending>} ⓘ  
  ► [[Prototype]]: Promise  
    [[PromiseState]]: "fulfilled"  
    [[PromiseResult]]: undefined  
  
  > getData  
  
< ▼ Promise {<pending>} ⓘ ←  
  ► [[Prototype]]: Promise  
    [[PromiseState]]: "fulfilled"  
    [[PromiseResult]]: "Donnée récupérée"
```

Après 3 secondes

Ainsi, l'appel de la fonction **resolve()** déplace l'objet de promesse vers l'état **fulfilled**. Si vous modifiez la valeur de la variable **completed** false et exécutez à nouveau le script :

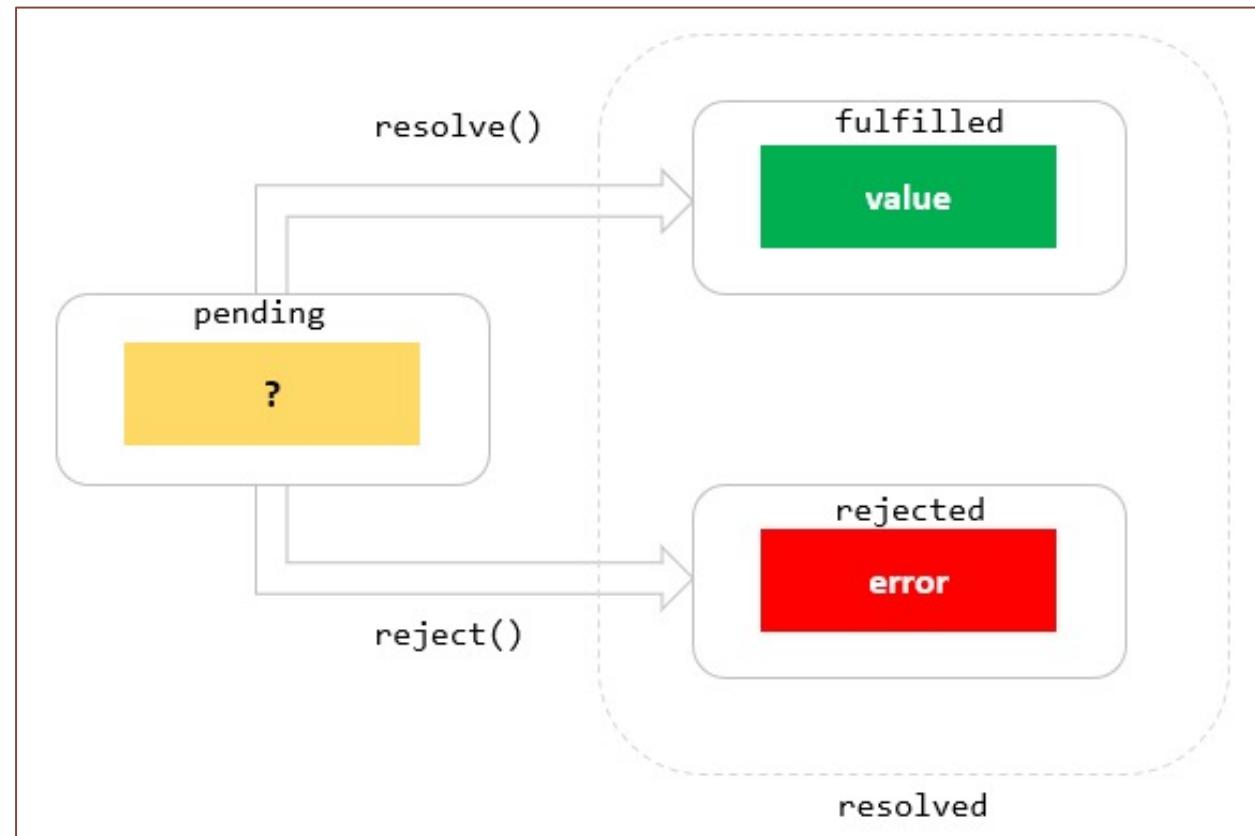
```
✖ Uncaught (in promise) Je n'ai pas pu récupérer la donnée  
  > getData  
  < ▼ Promise {<rejected>: "Je n'ai pas pu récupérer la donnée"} ⓘ  
    ► [[Prototype]]: Promise  
      [[PromiseState]]: "rejected"  
      [[PromiseResult]]: "Je n'ai pas pu récupérer la donnée"
```

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)

Créer une promesse : le constructeur Promise

L'image suivante illustre les états d'une promesse et l'effet de l'appel des fonctions `resolve()` et `reject()` :



03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)



Consommer une promesse : then, catch, finally

Pour planifier un rappel lorsque la promesse est résolue ou rejetée, vous appelez les méthodes de l' Promiseobjet : then(), catch(), et finally().

La méthode then()

La méthode then() est utilisée pour planifier l'exécution d'un rappel lorsque la promesse est résolue avec succès.

La méthode then() prend deux fonctions de rappel :

```
promiseObject.then(onFulfilled, onRejected);
```

Le callback **onFulfilled** est appelé si la promesse est remplie. Le callback **onRejected** est appelé lorsque la promesse est rejetée.

La fonction suivante renvoie un objet Promise :

```
function makePromise(completed){  
return new Promise(function (resolve, reject) {  
    setTimeout(() => {  
        if (completed) {  
            resolve("Donnée récupérée");  
        } else {  
            reject("Je n'ai pas pu récupérer la donnée");  
        }  
    }, 3000);  
});  
}
```

Et ce qui suit appelle la fonction makePromise()
et invoque la méthode then() :

```
let getData = makePromise(true);  
getData.then(  
    success=>console.log(success),  
    reason=>console.log(reason)  
)
```

Il est possible de programmer un rappel pour traiter uniquement le cas traité ou rejeté. Ce qui suit exécute le cas rempli :

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)

Consommer une promesse : then, catch, finally

La méthode catch()

Si vous souhaitez programmer un rappel à exécuter lorsque la promesse est rejetée, vous pouvez utiliser la méthode catch() de l'objet Promise :

```
getData.catch(
    reason=>console.log(reason)
);
```

En interne, la méthode catch() appelle la méthode then(undefined, onRejected).

La méthode finally()

Parfois, vous souhaitez exécuter le même morceau de code, que la promesse soit tenue ou rejetée.

```
getData
  .then(
    (success) => {
      console.log(success);
      createApp();
    }
  ).catch(
    (reason) => {
      console.log(reason);
      createApp();
    }
)
```

La fonction calculer()
est répétée deux fois
dans ce code



```
getData
  .then(success => console.log(success))
  .catch(reason => console.log(reason))
  .finally(() => createApp());
```

Ici, La fonction
calculer() n'est pas
répétée

03 – UTILISER DES FONCTIONS

Appels asynchrones (callBack, Promise)

Consommer une promesse : then, catch, finally

La méthode catch()

Si vous souhaitez programmer un rappel à exécuter lorsque la promesse est rejetée, vous pouvez utiliser la méthode catch() de l'objet Promise :

```
getData.catch(
    reason=>console.log(reason)
);
```

En interne, la méthode catch() appelle la méthode then(undefined, onRejected).

La méthode finally()

Parfois, vous souhaitez exécuter le même morceau de code, que la promesse soit tenue ou rejetée.

```
getData
  .then(
    (success) => {
      console.log(success);
      createApp();
    }
  ).catch(
    (reason) => {
      console.log(reason);
      createApp();
    }
)
```

La fonction calculer()
est répétée deux fois
dans ce code



```
getData
  .then(success => console.log(success))
  .catch(reason => console.log(reason))
  .finally(() => createApp());
```

Ici, La fonction
calculer() n'est pas
répétée



CHAPITRE 3

UTILISER DES FONCTIONS

1. Fonctions
2. Expressions lambdas
3. Appels asynchrones (callBack, Promise)
4. Gestion des exceptions

03 – UTILISER DES FONCTIONS

Gestion des exceptions

Le bloc try ... catch

L'instruction try vous permet de définir un bloc de code à tester pour les erreurs pendant son exécution.

L'instruction catch vous permet de définir un bloc de code à exécuter, si une erreur se produit dans le bloc try.

Les instructions JavaScript try et catch viennent par paires :

Syntaxe

```
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}
```

Exemple

On affiche "Réussi" si la note est supérieure ou égal à 10 :

```
if (note >= 10) {  
    document.write("Réussi");  
}
```

03 – UTILISER DES FONCTIONS

Gestion des exceptions



Le bloc try ... catch

JavaScript renvoie des erreurs

Lorsqu'une erreur se produit, JavaScript s'arrête normalement et génère un message d'erreur.

Le terme technique pour cela est : JavaScript lèvera une exception (lancera une erreur) .

L'instruction throw

L'instruction throw vous permet de créer une erreur personnalisée.

Techniquement, vous pouvez lever une exception (lancer une erreur) .

L'exception peut être un JavaScript String, un Number, un Boolean ou un Object:

```
throw "Too big";    // throw a text  
throw 500;
```

Si vous utilisez throw avec try et catch, vous pouvez contrôler le déroulement du programme et générer des messages d'erreur personnalisés.

03 – UTILISER DES FONCTIONS

Gestion des exceptions



Exemple de validation d'entrée

Cet exemple examine l'entrée. Si la valeur est erronée, une exception (err) est levée.

L'exception (err) est interceptée par l'instruction catch et un message d'erreur personnalisé s'affiche :

```
<!DOCTYPE html>
<html>
<body>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test
Input</button>
<p id="p01"></p>

<script>
</script>

</body>
</html>
```

```
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
```

03 – UTILISER DES FONCTIONS

Gestion des exceptions

La déclaration finally

L'instruction finally vous permet d'exécuter du code, après try and catch, quel que soit le résultat :

Syntaxe

```
try {  
    Bloc de code à essayer d'exécuter  
}  
catch(err) {  
    Bloc de code qui récupère l'erreur  
}  
finally {  
    Bloc à executer quel que soit le  
résultat  
}
```

```
function myFunction() {  
    const message = document.getElementById("p01");  
    message.innerHTML = "";  
    let x = document.getElementById("demo").value;  
    try {  
        if(x == "") throw "is empty";  
        if(isNaN(x)) throw "is not a number";  
        x = Number(x);  
        if(x > 10) throw "is too high";  
        if(x < 5) throw "is too low";  
    }  
    catch(err) {  
        message.innerHTML = "Error: " + err + ".";  
    }  
    finally {  
        document.getElementById("demo").value = "";  
    }  
}
```

03 – UTILISER DES FONCTIONS

Gestion des exceptions

L'objet d'erreur

JavaScript a un objet d'erreur intégré qui fournit des informations sur l'erreur lorsqu'une erreur se produit.

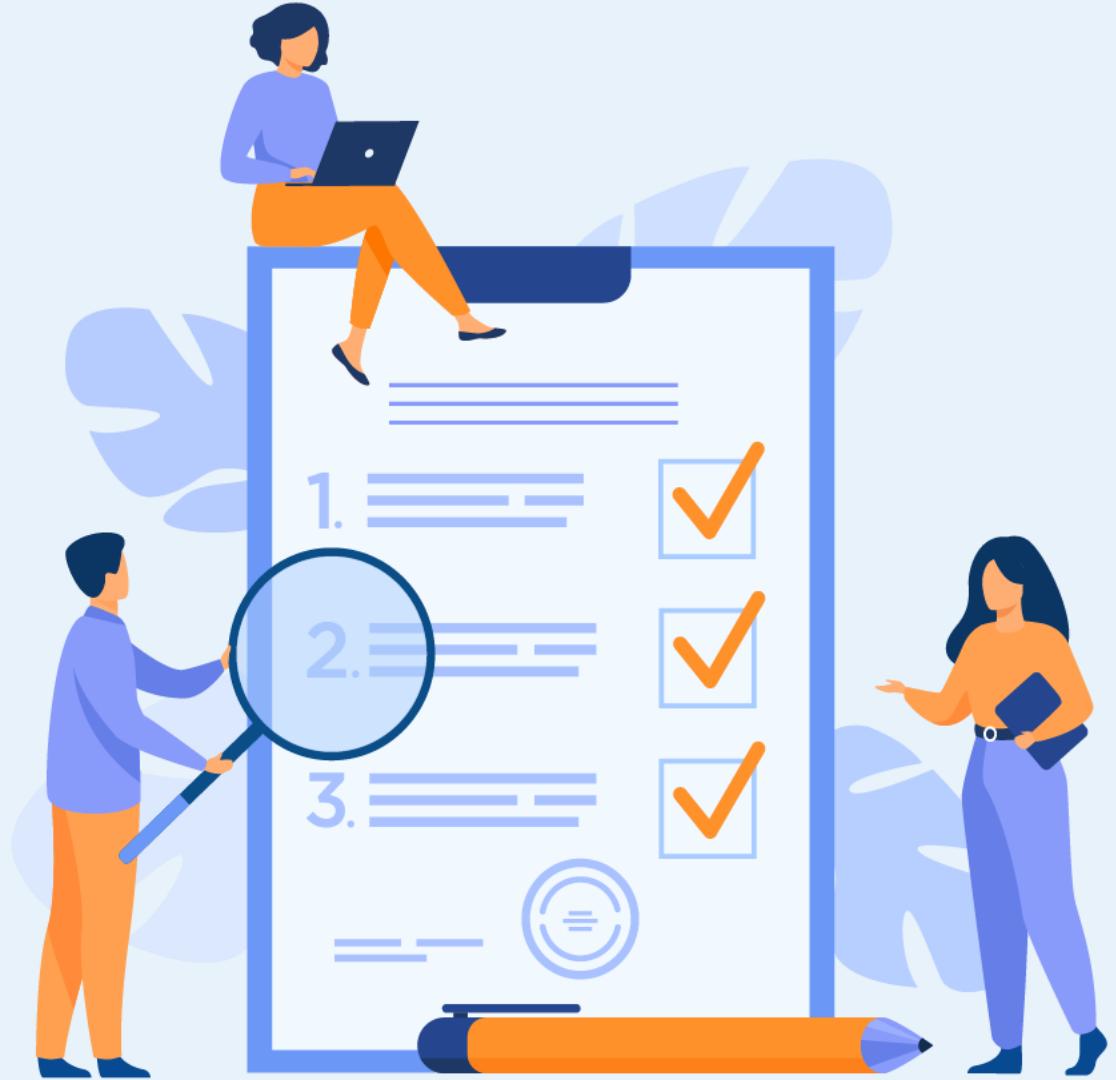
L'objet error fournit deux propriétés utiles : **name** et **message**.

Valeurs de nom d'erreur

Nom de l'erreur	La description
EvalError	Une erreur s'est produite dans la fonction eval()
RangeError	Un nombre "hors limites" s'est produit
ReferenceError	Une référence illégale s'est produite
SyntaxError	Une erreur de syntaxe s'est produite
TypeError	Une erreur de type s'est produite
URIError	Une erreur dans encodeURI() s'est produite

Exemple

```
let x = 5;
try {
  x = y + 1;    // y cannot be used (referenced)
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```



CHAPITRE 4

MANIPULER LES OBJETS

Ce que vous allez apprendre dans ce chapitre :

- Création d'objet
- Manipulation d'objet
- Manipulation des objets natifs
- Manipulation JSON



12 heures



CHAPITRE 4

MANIPULER LES OBJETS

1. Création d'objet
2. Manipulation d'objet
3. Manipulation des objets natifs
4. Manipulation JSON

04 – MANIPULER LES OBJETS

Création d'objet



Vous pouvez déclarer un objet directement dans une variable avec des propriétés et des méthodes,

Littéral

```
var telephone = {  
    name: 'Motorola',  
    price: 400,  
    stock: 200,  
    ref: 'moto z',  
    checkStock: function() {  
        if (this.stock > 0) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

Constructeur

```
function Telephone(name, price, stock, ref) {  
    this.name = name;  
    this.price = price;  
    this.stock = stock;  
    this.ref = ref;  
    this.checkStock = function() {  
        if (this.stock > 0) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}  
  
var _motoZ = new Telephone("Motorola", 400, 200, "Moto Z");  
var _xiaomiMiMax = new Telephone("Xiaomi", 200, 20, "Mi Max");
```

Lorsque vous créer un objet par l'intermédiaire d'un constructeur, vous utilisez une déclaration de fonction, vous pourrez par la suite créer plusieurs fois le même objet dans des variables différentes.



CHAPITRE 4

MANIPULER LES OBJETS

1. Création d'objet
2. Manipulation d'objet
3. Manipulation des objets natifs
4. Manipulation JSON

04 – MANIPULER LES OBJETS

Création d'objet



Ajouter/modifier des propriétés ou des méthodes

Une fois que vous avez créer un objet, vous pouvez également ajouter d'autres propriétés ou méthodes à celle-ci.
la syntaxe convient aux deux types de notations.

```
telephone.name = 'Oneplus';
```

Supprimer une propriété ou une fonction

Pour supprimer une propriété, vous devez indiquer l'objet suivi de la propriété devant le mot clé **delete** :

```
delete telephone.name;
```

Itérer sur les propriétés d'un objet à l'aide d'une boucle for...in

```
for (const key in telephone) {  
    console.log(key);  
}
```



CHAPITRE 4

MANIPULER LES OBJETS

1. Création d'objet
2. Manipulation d'objet
3. Manipulation des objets natifs
4. Manipulation JSON

04 – MANIPULER LES OBJETS

Manipulation des objets natifs



Les tableaux

Un tableau JS est un objet qui hérite de l'objet global standard `Array`, qui lui même hérite l'objet fondamental `Object`.

Pour une définition plus accessible, on peut dire qu'un tableau contient une liste d'éléments, qu'ils soient du même type ou non.

Créer un tableau

En JavaScript, un tableau vide peut se créer de 2 manières :

```
let tableau = new Array; // Ici c'est explicite, on crée une instance de l'objet Array  
let tableau = []; // Mais on peut aussi le déclarer de manière littérale avec []
```

Bien-sûr, on peut aussi créer des tableaux avec des valeurs initiales :

```
let tableau = new Array(5, 10, 15, 20);  
let tableau = ['A', 'B', 'C'];
```

Il est aussi possible d'initialiser un tableau avec un certain nombre d'éléments vides, il suffit dans ce cas de mettre un entier en paramètre de `Array` :

```
let tableau = new Array(3);
```

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Accéder aux éléments du tableau

Maintenant que nous avons un tableau, on va avoir besoin de récupérer les éléments dedans. Pour cela, on peut utiliser l'indice. Attention, pour rappel les indices d'un tableau débute à 0 et non à 1 :

```
let tableau = ['A', 'B', 'C'];
console.log(tableau[1]); // Retourne 'B'
```

A l'inverse, on peut avoir besoin de récupérer l'indice à partir de l'élément associé. C'est la méthode `indexOf()` qui fait cela :

```
let tableau = ['A', 'B', 'C'];
console.log(tableau.indexOf('C')); // Retourne 2
```

`indexOf()` a l'avantage d'avoir un deuxième paramètre optionnel, qui permet de choisir l'indice à partir duquel on débute la recherche. Par défaut, ce deuxième paramètre est à 0, mais on peut le modifier de cette manière :

```
let tableau = ['A', 'B', 'C', 'B'];
console.log(tableau.indexOf('B'));    // Retourne 1, L'indice du premier B trouvé
console.log(tableau.indexOf('B', 2)); // Retourne 3, L'indice du premier B trouvé après l'indice 2
```

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Récupérer la taille un tableau

Pour obtenir le nombre d'éléments d'un tableau, c'est très simple, on utilise la propriété `length` de l'objet `Array` :

```
let tableau = ['A', 'B', 'C'];
let nbElements = tableau.length;
console.log(nbElements);
// Retourne 3 (et non 2, ici length compte le nombre d'éléments, il ne se passe pas sur les indices)
```

Parcourir un tableau

Pour parcourir le contenu d'un tableau, l'objet `Array` dispose de sa propre méthode de boucle dans son prototype.

Il s'agit de `forEach()`, et permet d'exécuter une fonction (et donc des instructions) pour chaque élément du tableau. Par exemple :

```
let tableau = ['A', 'B'];
tableau.forEach(function(element) {
  console.log(element);
});
// Retourne :
// 'A';
// 'B';
```

Vous remarquerez que l'on passe en paramètre de la fonction l'élément, ce qui nous permet de le récupérer et de l'utiliser dans nos instructions.

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Ajouter un élément dans un tableau

Si l'on souhaite ajouter un élément, on utilise la méthode **push**

```
let tableau = ['A', 'B'];
tableau.push('C');
console.log(tableau); // La manière objet, Retourne ['A', 'B', 'C']
```

Bien sûr, on peut ajouter n'importe quel type d'élément à un tableau : Un nombre, un booléen, une chaîne de caractère, un objet et même un autre tableau. Il n'est pas nécessaire que tous les éléments d'un tableau soient du même type :

```
let tableau = ['A', 'B'];
tableau.push(22);
console.log(tableau);
// Retourne ['A', 'B', 22];
tableau.push( {id: 1, name: 'Nom'} );
console.log(tableau);
// Retourne ['A', 'B', 22, {id: 1, name: 'Nom'}];
```

Comme vous le voyez, la méthode **push()** ajoute notre élément à la fin du tableau. Si vous souhaitez l'ajouter au début, vous pouvez utiliser **unshift()** :

```
let tableau = ['A', 'B'];
tableau.unshift(22);
console.log(tableau);
// Retourne [22, 'A', 'B'];
```

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Modifier un élément du tableau

Pour modifier la valeur d'un élément, on peut directement utiliser son indice :

```
let tableau = ['A', 'B', 'C'];
tableau[1] = 'D';
console.log(tableau); // Retourne ['A', 'D', 'C'];
```

Supprimer un élément d'un tableau

Si vous avez besoin de supprimer le dernier élément d'un tableau, `pop()` est fait pour vous :

```
let tableau = ['A', 'B', 'C'];
tableau.pop();
console.log(tableau); // Retourne ['A', 'B'];
```

Même principe avec `shift()` pour supprimer le premier élément du tableau :

```
let tableau = ['A', 'B', 'C'];
tableau.shift();
console.log(tableau); // Retourne ['B', 'C'];
```

Vous utilisez également la méthode `splice()` pour supprimer un élément dans une position spécifique

```
let tableau = ['A', 'B', 'C'];
tableau.splice(1,1);
console.log(tableau); // Retourne ['B', 'C'];
```

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Trier un tableau

Il existe plusieurs méthodes de Array pour vous permettre de trier vos tableaux. Si vous souhaitez les classer par ordre alphabétique, utilisez sort() :

```
let tableau = ['E', 'A', 'D'];
tableau.sort();
console.log(tableau); // Retourne ['A', 'D', 'E']
```

Si on souhaite simplement retourner le tableau en ayant les derniers éléments en premier, c'est reverse() qu'il faut utiliser :

```
let tableau = ['A', 'B', 'C'];
tableau.reverse();
console.log(tableau); // Retourne ['C', 'B', 'A'];
```

Recherche un élément dans le tableau

Souvent, on a besoin d'un ou plusieurs éléments de notre tableau qui remplissent une condition.

Une première approche est d'utiliser la méthode **findIndex()** qui permet de remonter l'indice du premier élément de notre tableau qui rempli une condition. Par exemple :

```
function findC(element) {
  return element === 'C';
}
let tableau = ['A', 'B', 'C', 'D', 'C'];
tableau.findIndex(findC); // Retourne 2, L'indice de notre premier C
```

Objet Date JavaScript

L'objet date JavaScript peut être utilisé pour obtenir l'année, le mois et le jour. Vous pouvez afficher une minuterie sur la page Web à l'aide de l'objet date JavaScript.

Vous pouvez utiliser différents constructeurs de date pour créer un objet date. Il fournit des méthodes pour obtenir et régler le jour, le mois, l'année, l'heure, les minutes et les secondes.

Constructeur

Vous pouvez utiliser 4 variantes du constructeur Date pour créer un objet date.

- Date()
- Date (millisecondes)
- Date(chaîne de date)
- Date (année, mois, jour, heures, minutes, secondes, millisecondes)

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Méthodes de date JavaScript

Méthodes	Description
<code>getDate()</code>	Il renvoie la valeur entière comprise entre 1 et 31 qui représente le jour de la date spécifiée sur la base de l'heure locale.
<code>getDay()</code>	Il renvoie la valeur entière comprise entre 0 et 6 qui représente le jour de la semaine sur la base de l'heure locale.
<code>getFullYear()</code>	Il renvoie la valeur entière qui représente l'année sur la base de l'heure locale.
<code>getHours()</code>	Il renvoie la valeur entière comprise entre 0 et 23 qui représente les heures sur la base de l'heure locale.
<code>getMilliseconds()</code>	Il renvoie la valeur entière comprise entre 0 et 999 qui représente les millisecondes sur la base de l'heure locale.
<code>getMinutes()</code>	Il renvoie la valeur entière comprise entre 0 et 59 qui représente les minutes sur la base de l'heure locale.
<code>getMonth()</code>	Il renvoie la valeur entière comprise entre 0 et 11 qui représente le mois sur la base de l'heure locale.
<code>getSeconds()</code>	Il renvoie la valeur entière comprise entre 0 et 60 qui représente les secondes sur la base de l'heure locale.
<code>getUTCDate()</code>	Il renvoie la valeur entière comprise entre 1 et 31 qui représente le jour de la date spécifiée sur la base de l'heure universelle.
<code>getUTCDay()</code>	Il renvoie la valeur entière comprise entre 0 et 6 qui représente le jour de la semaine sur la base de l'heure universelle.
<code>getUTCFullYear()</code>	Il renvoie la valeur entière qui représente l'année sur la base du temps universel.
<code>getUTCHours()</code>	Il renvoie la valeur entière comprise entre 0 et 23 qui représente les heures sur la base du temps universel.
<code>getUTCMilliseconds()</code>	Il renvoie la valeur entière comprise entre 0 et 999 qui représente les millisecondes sur la base du temps universel.
<code>getUTCMonth()</code>	Il renvoie la valeur entière comprise entre 0 et 11 qui représente le mois sur la base du temps universel.
<code>getUTCSeconds()</code>	Il renvoie la valeur entière comprise entre 0 et 60 qui représente les secondes sur la base du temps universel.
<code> setDate()</code>	Il définit la valeur du jour pour la date spécifiée sur la base de l'heure locale.
<code> setDay()</code>	Il définit le jour particulier de la semaine sur la base de l'heure locale.
<code> setFullYear()</code>	Il définit la valeur de l'année pour la date spécifiée sur la base de l'heure locale.

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Méthodes de date JavaScript

Méthodes	Description
<code>setHours()</code>	Il définit la valeur de l'heure pour la date spécifiée sur la base de l'heure locale.
<code>setMilliseconds()</code>	Il définit la valeur en millisecondes pour la date spécifiée sur la base de l'heure locale.
<code>setMinutes()</code>	Il définit la valeur des minutes pour la date spécifiée sur la base de l'heure locale.
<code>setMonth()</code>	Il définit la valeur du mois pour la date spécifiée sur la base de l'heure locale.
<code>setSeconds()</code>	Il définit la deuxième valeur pour la date spécifiée sur la base de l'heure locale.
<code>setUTCDate()</code>	Il définit la valeur du jour pour la date spécifiée sur la base du temps universel.
<code>setUTCDay()</code>	Il fixe le jour particulier de la semaine sur la base du temps universel.
<code>setUTCFullYear()</code>	Il définit la valeur de l'année pour la date spécifiée sur la base du temps universel.
<code>setUTCHours()</code>	Il définit la valeur de l'heure pour la date spécifiée sur la base du temps universel.
<code>setUTCMilliseconds()</code>	Il définit la valeur en millisecondes pour la date spécifiée sur la base du temps universel.
<code>setUTCMinutes()</code>	Il définit la valeur des minutes pour la date spécifiée sur la base du temps universel.
<code>setUTCMonth()</code>	Il définit la valeur du mois pour la date spécifiée sur la base du temps universel.
<code>setUTCSeconds()</code>	Il définit la deuxième valeur pour la date spécifiée sur la base du temps universel.
<code>toDateString()</code>	Il renvoie la partie date d'un objet Date.
<code>toISOString()</code>	Il renvoie la date sous la forme d'une chaîne de format ISO.
<code> toJSON()</code>	Il renvoie une chaîne représentant l'objet Date. Il sérialise également l'objet Date lors de la sérialisation JSON.
<code>toString()</code>	Il renvoie la date sous forme de chaîne.
<code>toTimeString()</code>	Il renvoie la partie heure d'un objet Date.
<code>toUTCString()</code>	Il convertit la date spécifiée sous forme de chaîne en utilisant le fuseau horaire UTC.
<code>valueOf()</code>	Il renvoie la valeur primitive d'un objet Date.

04 – MANIPULER LES OBJETS

Manipulation des objets natifs



Méthodes de l'objet Math

Méthodes	Description
<code>abs()</code>	Il renvoie la valeur absolue du nombre donné.
<code>acos()</code>	Il renvoie l'arc cosinus du nombre donné en radians.
<code>asin()</code>	Il renvoie l'arc sinus du nombre donné en radians.
<code>atan()</code>	Il renvoie l'arc tangente du nombre donné en radians.
<code>ceil()</code>	Il renvoie une plus petite valeur entière, supérieure ou égale au nombre donné.
<code>cos()</code>	Il renvoie le cosinus du nombre donné.
<code>exp()</code>	Il renvoie la forme exponentielle du nombre donné.
<code>floor()</code>	Il renvoie la plus grande valeur entière, inférieure ou égale au nombre donné.
<code>hypot()</code>	Il renvoie la racine carrée de la somme des carrés de nombres donnés.
<code>log()</code>	Il renvoie le logarithme népérien d'un nombre.
<code>max()</code>	Il renvoie la valeur maximale des nombres donnés.
<code>min()</code>	Il renvoie la valeur minimale des nombres donnés.
<code>pow()</code>	Il renvoie la valeur de la base à la puissance de l'exposant.
<code>random()</code>	Il renvoie un nombre aléatoire compris entre 0 (inclus) et 1 (exclusif).
<code>round()</code>	Il renvoie la valeur entière la plus proche du nombre donné.
<code>sin()</code>	Il renvoie le sinus du nombre donné.
<code>sqrt()</code>	Il renvoie la racine carrée du nombre donné
<code>tan()</code>	Il renvoie la tangente du nombre donné.
<code>trunc()</code>	Il renvoie une partie entière du nombre donné.

04 – MANIPULER LES OBJETS

Manipulation des objets natifs



Les regex

Une expression régulière est une séquence de caractères qui forme un modèle de recherche .

Lorsque vous recherchez des données dans un texte, vous pouvez utiliser ce modèle de recherche pour décrire ce que vous recherchez.

Une expression régulière peut être un caractère unique ou un modèle plus compliqué.

Les expressions régulières peuvent être utilisées pour effectuer tous les types d'opérations de recherche de texte et de remplacement de texte .

Syntaxe

/pattern/modifiers;

Exemple expliqué :

/ofppt/i est une expression régulière.

ofppt est un modèle (à utiliser dans une recherche).

i est un modificateur (modifie la recherche pour qu'elle soit insensible à la casse).

04 – MANIPULER LES OBJETS

Manipulation des objets natifs



Les regex - Utilisation des méthodes de chaîne

En JavaScript, les expressions régulières sont souvent utilisées avec les deux méthodes de chaîne : search() et replace().

La méthode search() utilise une expression pour rechercher une correspondance et renvoie la position de la correspondance.

La méthode replace() renvoie une chaîne modifiée où le motif est remplacé..

Exemple1

Utilisez une chaîne pour rechercher "ofppt" dans une chaîne :

```
let text = "Visit ofppt!";
let n = text.search("ofppt"); //renvoie 6
console.log(n)
```

Exemple2

La méthode replace() remplace une valeur spécifiée par une autre valeur dans une chaîne :

```
let text = "Visit Microsoft!";
let result = text.replace("Microsoft", "ofppt");
```

Utilisez String replace() avec une expression régulière

Exemple3

Utilisez une expression régulière insensible à la casse pour remplacer Microsoft par ofppt dans une chaîne :

```
let text = "Visit Microsoft!";
let result = text.replace(/microsoft/i, "OFPPT");
```

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Modificateurs d'expressions régulières

Les modificateurs peuvent être utilisés pour effectuer des recherches plus globales insensibles à la casse :

Modificateur	Description
i	Effectuer une correspondance insensible à la casse
g	Effectuer une correspondance globale (trouver toutes les correspondances plutôt que de s'arrêter après la première)
m	Effectuer une correspondance multiligne

Modèles d'expressions régulières

Les crochets sont utilisés pour rechercher une plage de caractères :

Expression	Description
[abc]	Trouvez l'un des caractères entre les crochets
[0-9]	Trouvez l'un des chiffres entre les crochets
(x y)	Trouvez l'une des alternatives séparées par

04 – MANIPULER LES OBJETS

Manipulation des objets natifs



Modèles d'expressions régulières

Les métacaractères sont des caractères ayant une signification particulière :

Expression	Description
\d	Trouver un chiffre
\s	Trouver un caractère d'espacement
\b	Trouvez une correspondance au début d'un mot comme celui-ci : \bWORD, ou à la fin d'un mot comme celui-ci : WORD\b
\uxxxx	Trouvez le caractère Unicode spécifié par le nombre hexadécimal xxxx

Les quantificateurs définissent les quantités :

Quantifier	Description
n+	Correspond à toute chaîne contenant au moins un n
n*	Correspond à toute chaîne contenant zéro ou plusieurs occurrences de n
n?	Correspond à toute chaîne contenant zéro ou une occurrence de n

04 – MANIPULER LES OBJETS

Manipulation des objets natifs

Utilisation de l'objet RegExp

En JavaScript, l'objet RegExp est un objet d'expression régulière avec des propriétés et des méthodes prédéfinies.

Utilisation de test()

La méthode test() est une méthode d'expression RegExp.

Il recherche un motif dans une chaîne et renvoie vrai ou faux, selon le résultat.

L'exemple suivant recherche une chaîne pour le caractère "e":

Exemple

```
const pattern = /e/;  
pattern.test("The best things in life are free!"); //Renvoie true
```

Utiliser exec()

La méthode exec() est une méthode d'expression RegExp.

Il recherche dans une chaîne un modèle spécifié et renvoie le texte trouvé en tant qu'objet.

Si aucune correspondance n'est trouvée, elle renvoie un objet vide (null) .

L'exemple suivant recherche une chaîne pour le caractère "e":

```
/e/.exec("The best things in life are free!");
```



CHAPITRE 4

MANIPULER LES OBJETS

1. Création d'objet
2. Manipulation d'objet
3. Manipulation des objets natifs
4. Manipulation JSON

04 – MANIPULER LES OBJETS

Manipulation JSON



JSON

Une utilisation courante de JSON est d'échanger des données vers/depuis un serveur Web.

Lors de la réception de données d'un serveur Web, les données sont toujours une chaîne.

Analysez les données avec `JSON.parse()`, et les données deviennent un objet JavaScript

Exemple - Analyse JSON

Imaginez que nous recevions ce texte d'un serveur Web :

```
const obj = {name: "Hassan", age: 30, city: "Rabat"};
```

Utilisez la fonction JavaScript `JSON.parse()` pour convertir du texte en objet JavaScript :

```
const obj = JSON.parse('{"name": "John", "age": 30, "city": "New York"}');
```

Utilisez l'objet JavaScript dans votre page :

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = obj.name;
</script>
```

04 – MANIPULER LES OBJETS

Manipulation JSON



Tableau au format JSON

Lorsque vous utilisez le `JSON.parse()` sur un JSON dérivé d'un tableau, la méthode renvoie un tableau JavaScript, au lieu d'un objet JavaScript.

```
const text = '[ "Ford", "BMW", "Audi", "Fiat"]';
const myArr = JSON.parse(text);
```

Stringifier un objet JavaScript

Imaginez que nous ayons cet objet en JavaScript : `const obj = {name: "Hassan", age: 30, city: "Rabat"};`

Utilisez la fonction JavaScript `JSON.stringify()` pour le convertir en chaîne. `const myJSON = JSON.stringify(obj);`

Enregistrer des données

Lors de l'enregistrement des données, les données doivent être dans un certain format, et quel que soit l'endroit où vous choisissez de les stocker, le texte est toujours l'un des formats légaux.

JSON permet de stocker des objets JavaScript sous forme de texte.

```
// Storing data:
const myObj = {name: "John", age: 31, city: "New York"};
const myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
// Retrieving data:
let text = localStorage.getItem("testJSON");
let obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```



PARTIE 3

MANIPULER LES ÉLÉMENTS D'UNE PAGE AVEC DOM

Dans ce module, vous allez :

- Comprendre l'arbre DOM, les nœuds parents et enfants
- Connaître les bases de la manipulation du DOM en JavaScript
- Manipuler les éléments HTML



 30 heures



CHAPITRE 1

COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Ce que vous allez apprendre dans ce chapitre :

- Arbre DOM
- Objet Document
- Navigation dans le DOM (parentNode, childNodes, ...)

 10 heures



CHAPITRE 1

COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

1. Arbre DOM
2. Objet Document
3. Navigation dans le DOM (`parentNode`, `childNodes`, ...)

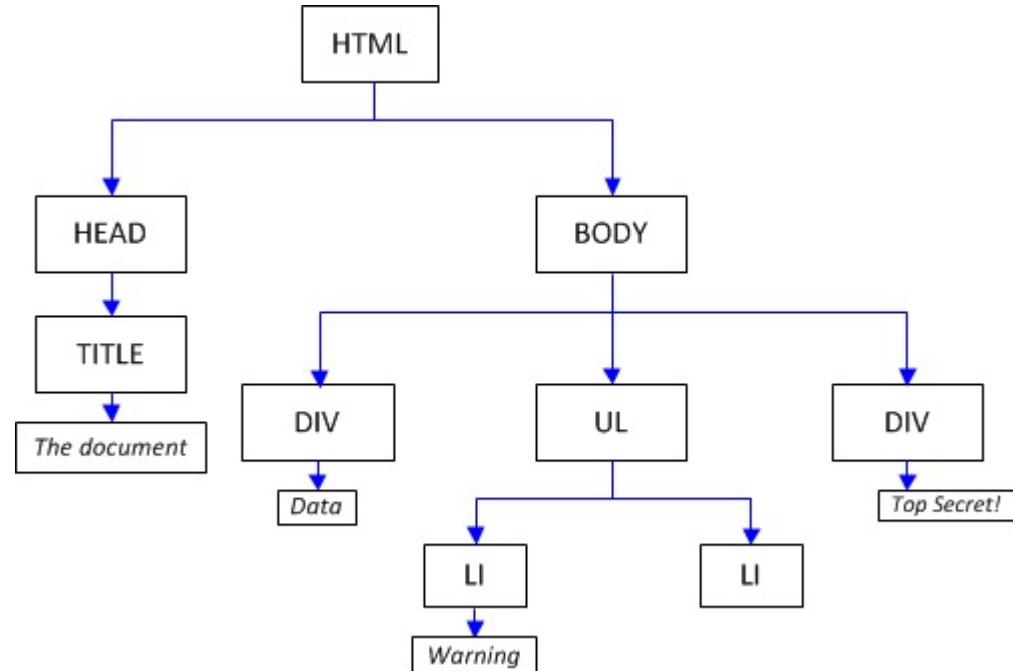
01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Arbre DOM



Lorsqu'une page Web est chargée, le navigateur crée un Document Objet Modèle de la page.

Le modèle HTML DOM est construit comme un arbre d' Objets :



01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Arbre DOM



Avec le modèle objet, JavaScript obtient toute la puissance dont il a besoin pour créer du HTML dynamique :

- JavaScript peut modifier tous les éléments HTML de la page
- JavaScript peut modifier tous les attributs HTML de la page
- JavaScript peut changer tous les styles CSS de la page
- JavaScript peut supprimer les éléments et attributs HTML existants
- JavaScript peut ajouter de nouveaux éléments et attributs HTML
- JavaScript peut réagir à tous les événements HTML existants dans la page
- JavaScript peut créer de nouveaux événements HTML dans la page

Le DOM HTML est un modèle objet standard et une interface de programmation pour HTML. Il définit :

- Les éléments HTML comme objets
- Les propriétés de tous les éléments HTML
- Les méthodes pour accéder à tous les éléments HTML
- Les événements pour tous les éléments HTML
- En d'autres termes : le DOM HTML est une norme sur la façon d'obtenir, de modifier, d'ajouter ou de supprimer des éléments HTML.



CHAPITRE 1

COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

1. Arbre DOM
2. Objet Document
3. Navigation dans le DOM (`parentNode`, `childNodes`, ...)

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Objet Document



L'objet document représente votre page Web.

Si vous souhaitez accéder à n'importe quel élément d'une page HTML, vous commencez toujours par accéder à l'objet document.

Vous trouverez ci-dessous quelques exemples d'utilisation de l'objet document pour accéder au HTML et le manipuler.

Recherche d'éléments HTML

Method	Description
<code>document.getElementById(id)</code>	Trouver un élément par élément ID
<code>document.getElementsByTagName(name)</code>	Trouver des éléments par nom de balise
<code>document.getElementsByClassName(name)</code>	Trouver des éléments par nom de classe

Modification des éléments HTML

Propriété	Description
<code>element.innerHTML = new html content</code>	Changer la valeur de l'innerHTML
<code>element.attribute = new value</code>	Changer l'attribut d'un élément
<code>element.style.property = new style</code>	Changer le style d'un élément

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Objet Document



Ajout et suppression d'éléments

Method	Description
<code>document.createElement(element)</code>	Créer un élément HTML
<code>document.removeChild(element)</code>	Supprimer un élément HTML
<code>document.appendChild(element)</code>	Ajouter un élément HTML enfant
<code>document.replaceChild(new, old)</code>	Remplacer un élément HTML
<code>document.write(text)</code>	Ecrire dans un document HTML

Ajout de gestionnaires d'événements

Méthode	Description
<code>document.getElementById(id).onclick = function(){code}</code>	Ajouter un événement de clic

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Objet Document



Ajout et suppression d'éléments

Method	Description
<code>document.createElement(element)</code>	Créer un élément HTML
<code>document.removeChild(element)</code>	Supprimer un élément HTML
<code>document.appendChild(element)</code>	Ajouter un élément HTML enfant
<code>document.replaceChild(new, old)</code>	Remplacer un élément HTML
<code>document.write(text)</code>	Ecrire dans un document HTML

Ajout de gestionnaires d'événements

Méthode	Description
<code>document.getElementById(id).onclick = function(){code}</code>	Ajouter un événement de clic



CHAPITRE 1

COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

1. Arbre DOM
2. Objet Document
3. Navigation dans le DOM (`parentNode`, `childNodes`, ...)

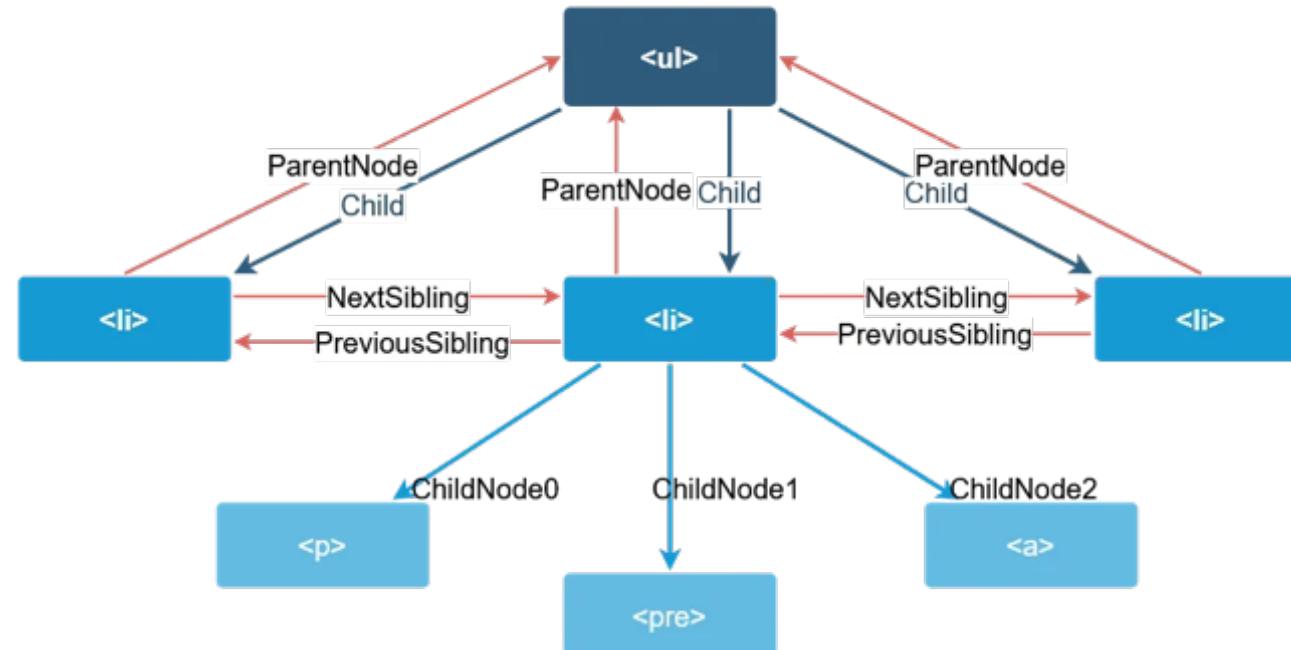
01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM

Relations de nœuds

Les nœuds de l'arborescence des nœuds ont une relation hiérarchique les uns avec les autres. Les termes parent, enfant et frère sont utilisés pour décrire les relations.

- Dans une arborescence de nœuds, le nœud supérieur est appelé racine (ou nœud racine)
- Chaque nœud a exactement un parent, sauf la racine (qui n'a pas de parent)
- Un nœud peut avoir plusieurs enfants
- Les frères et sœurs (frères ou sœurs) sont des nœuds avec le même parent



01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Navigation entre les nœuds

Voyons les différentes manières fournies par JavaScript pour se déplacer dans l'arborescence DOM dans une direction spécifique :

1. Descendre dans l'arborescence DOM : Il existe 6 différentes manières JavaScript de déplacer et d'obtenir des nœuds enfants

1. **firstChild** - Retourne le premier enfant de l'élément
2. **firstElementChild** - Retourne le premier élément enfant du parent
3. **lastChild** - Retourne le dernier enfant de l'élément
4. **lastElementChild** - Retourne le dernier élément enfant du parent
5. **childNodes** - Renvoie tous les enfants de l'élément sous la forme d'une collection de tableaux
6. **children** - Renvoie tous les enfants qui sont des éléments sous la forme d'une collection de tableaux

2. Remonter dans l'arborescence DOM : Il existe 2 manières JavaScript différentes de déplacer et d'accéder aux nœuds parents

1. **parentNode** - Renvoie le nœud parent de l'élément
2. **parentElement** - Renvoie le nœud de l'élément parent de l'élément

3. Déplacer latéralement dans l'arborescence : il existe 4 façons différentes de se déplacer et d'accéder aux nœuds frères en JavaScript

1. **nextSibling** - Renvoie le nœud frère qui est le prochain enfant de son parent
2. **nextElementSibling** - Renvoie l'élément frère qui est le prochain enfant de son parent
3. **previousSibling** - Renvoie le nœud frère qui est un enfant précédent de son parent
4. **previousElementSibling** - Renvoie l'élément frère qui est un enfant précédent de son parent

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

firstChild

En utilisant la propriété `firstChild` node sur un nœud, vous obtiendrez le premier nœud de l'élément. Le premier nœud n'est pas nécessairement un élément, il peut également contenir du texte ou un commentaire.

```
<div id="parent">
  <h1>This is heading</h1>
  <p>This is paragraph</p>
</div>

<script>
  const element = document.getElementById("parent");
  const first = element.firstChild;
  console.log(first); // text node
  console.log(first.nodeName); // output: #text
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

firstElementChild

Pour obtenir ce que nous attendions (premier enfant qui est un élément), utilisez le firstElementChild, il renvoie le premier enfant du parent qui est un élément.

```
<div id="parent">
  <h1>This is heading</h1>
  <p>This is paragraph</p>
</div>

<script>
  var element = document.getElementById("parent");
  var firstelement = element.firstElementChild.nodeName;
  console.log(firstelement); // output: H1
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

lastChild

Maintenant, sélectionnons le dernier enfant du parent. Utilisez la propriété lastChild pour sélectionner le dernier enfant de l'élément parent.

Il renvoie null s'il n'y a pas d'enfant.

```
<div id="parent">
  <h1>This is heading</h1>
  <p>This is paragraph</p>
</div>

<script>
  var element = document.getElementById("parent");
  var last = element.lastChild.nodeName;
  console.log(last); // output: #text
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

lastElementChild

Identique à la propriété firstChild, le dernier enfant n'a pas besoin d'être un élément pour obtenir le dernier enfant qui est la propriété lastChild d'utilisation de l'élément .

Il renvoie null s'il n'y a pas d'enfant.

```
<div id="parent">
  <h1>This is heading</h1>
  <p>This is paragraph</p>
</div>

<script>
  var element = document.getElementById("parent");
  var lastElement = element.lastElementChild.nodeName;
  console.log(lastElement); // output: P
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

childNodes

La propriété childNodes d'un nœud renvoie une NodeList en direct des nœuds enfants de l'élément donné.

Les enfants reçoivent un index où 0 est un index du premier enfant. Dans l'exemple ci-dessous, tous les nœuds enfants sont enregistrés.

```
<div id="parent">
  <h1>This is heading</h1>
  <p>This is paragraph</p>
  <a href="#">link</a>
</div>

<script>
  const element = document.getElementById("parent");
  for (let i = 0; i < element.childNodes.length; i++) {
    console.log(element.childNodes[i]);
  }
</script>
```

Vous pouvez voir dans l'exemple ci-dessus que la propriété childNodes renvoie tous les types de nœud, y compris les nœuds '#text' et 'element'.

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

children

Pour obtenir uniquement les nœuds d' élément , utilisez la propriété children sur l'élément parent.

La propriété children de element renvoie HTMLCollection qui contient tous les éléments enfants de l'élément spécifié.

```
<div id="parent">
    <h1>This is heading</h1>
    <p>This is paragraph</p>
    <a href="#">link</a>
</div>

<script>
    var parent = document.getElementById("parent");
    var children = parent.children;

    for (let i = 0; i < children.length; i++) {
        document.write(children[i].nodeName + " - " + children[i].textContent + "<br>");
    }
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

parentNode

La propriété parentNode renvoie l'élément parent de l'élément appelant.

Si le parent n'existe pas ou si l'élément n'est pas attaché à l'arborescence DOM, il renvoie null.

```
<div>
  <h1 id="id1">This is heading</h1>
  <p>This is paragraph</p>
  <a href="#">link</a>
</div>

<script>
  const element = document.getElementById("id1");
  const parent = element.parentNode;
  document.write("Parent name - " + parent.nodeName);
  console.log(parent);
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

parentElement

parentElement renvoie également le parent de l'élément. La seule différence est que lorsque l'élément appelant est qu'il renvoie null tandis que parentNode renvoie document .

```
<div>
  <h1 id="id1">This is heading</h1>
  <p>This is paragraph</p>
  <a href="#">link</a>
</div>

<script>
  const element = document.getElementById("id1");
  const parent = element.parentElement;
  document.write("Parent element is - " + parent.nodeName);
  // parentNode vs parentElement
  console.log(document.documentElement.parentNode); // document
  console.log(document.documentElement.parentElement); // null
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

nextSibling

Pour accéder au prochain frère, utilisez la propriété nextSibling.
Le prochain frère n'est pas nécessairement un nœud d'élément.

```
<h1 id="id1">This is heading</h1>
<p>This is paragraph</p>
<a href="#">link</a>
<br>

<script>
  const element = document.getElementById("id1");
  const next = element.nextSibling;
  document.write("next sibling node - " + next.nodeName);
  console.log(next); // text node
</script>
```

Vous pouvez voir dans l'exemple ci-dessus nextSibling renvoie le nœud '#text' car c'est le nœud suivant.

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

nextSibling

Pour accéder au prochain frère, utilisez la propriété nextSibling.
Le prochain frère n'est pas nécessairement un nœud d'élément.

```
<h1 id="id1">This is heading</h1>
<p>This is paragraph</p>
<a href="#">link</a>
<br>

<script>
  const element = document.getElementById("id1");
  const next = element.nextSibling;
  document.write("next sibling node - " + next.nodeName);
  console.log(next); // text node
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

nextElementSibling

Pour obtenir le nœud d'élément immédiatement suivant de l'élément appelant, utilisez la propriété nextElementSibling.

```
<h1 id="id1">This is heading</h1>
<p>This is paragraph</p>
<a href="#">link</a>

<script>
    var element = document.getElementById("id1");
    var nextelement = element.nextElementSibling;
    document.write("next sibling element - " + nextelement.nodeName);
    console.log(nextelement);
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

previousSibling

Pour obtenir le nœud précédent qui est un élément, utilisez la propriété previousSibling sur l'élément.

```
<h1>This is heading</h1>
<p id="id1">This is paragraph</p>
<a href="#">link</a>

<script>
    var element = document.getElementById("id1");
    var previousnode = element.previousSibling;
    document.write("previous sibling node - " + previousnode.nodeName);
    console.log(previousnode);
</script>
```

01 – COMPRENDRE L'ARBRE DOM, LES NŒUDS PARENTS ET ENFANTS

Navigation dans le DOM



Exemples

previousElementSibling

Pour obtenir le nœud précédent qui est un élément, utilisez la propriété previousElementSibling sur l'élément.

```
<h1>This is heading</h1>
<p id="id1">This is paragraph</p>
<a href="#">link</a>

<script>
    var element = document.getElementById("id1");
    var previouselement = element.previousElementSibling;
    document.write("previous sibling element - " + previouselement.nodeName);
    console.log(previouselement);
</script>
```



CHAPITRE 2

CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

Ce que vous allez apprendre dans ce chapitre :

- Sélecteurs (simples, multiples...)
- Modes d'Accès aux éléments



10 heures



CHAPITRE 2

CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

- Sélecteurs (simples, multiples...)
- Modes d'Accès aux éléments

02 – CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

Sélecteurs (simples, multiples...)



La méthode `querySelector()` renvoie le premier élément qui correspond à un ou plusieurs sélecteurs CSS spécifiés dans le document.

Remarque : La méthode `querySelector()` ne renvoie que le premier élément qui correspond aux sélecteurs spécifiés. Pour renvoyer toutes les correspondances, utilisez plutôt la méthode `querySelectorAll()`.

Si le sélecteur correspond à un identifiant dans un document qui est utilisé plusieurs fois (notez qu'un "identifiant" doit être unique dans une page et ne doit pas être utilisé plus d'une fois), il renvoie le premier élément correspondant.

Syntaxe

```
document.querySelector(CSS selectors)
```

Exemples

Obtenez le premier élément `<p>` dans le document :

```
document.querySelector("p");
```

Obtenez le premier élément `<p>` du document avec `class="example"`:

```
document.querySelector("p.example");
```

Changez le texte d'un élément avec `id="demo"`:

```
document.querySelector("#demo").innerHTML = "Hello World!";
```

02 – CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

Sélecteurs (simples, multiples...)



Exemples:

Obtenez le premier élément <p> dans le document où le parent est un élément <div>.

```
document.querySelector("div > p");
```

Obtenez le premier élément <a> dans le document qui a un attribut "target":

```
document.querySelector("a[target]");
```

Obtenez tous les éléments <p> du document et définissez la couleur d'arrière-plan du premier élément <p> (index 0) :

```
// Get all <p> elements in the document
var x = document.querySelectorAll("p");
// Set the background color of the first <p> element
x[0].style.backgroundColor = "red";
```

Obtenez tous les éléments <p> du document avec class="example", et définissez l'arrière-plan du premier élément <p> :

```
// Get all <p> elements in the document with class="example"
var x = document.querySelectorAll(".example");
// Set the background color of the first <p> element with class="example" (index 0)
x[0].style.backgroundColor = "red";
```

Découvrez combien d'éléments avec class="example" il y a dans le document (en utilisant la propriété length de l'objet NodeList) :

```
var x = document.querySelectorAll(".example").length;
```

02 – CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

Sélecteurs (simples, multiples...)



Exemples:

Définissez la couleur d'arrière-plan de tous les éléments du document avec class="example":

```
var x = document.querySelectorAll(".example");
var i;
for (i = 0; i < x.length; i++) {
  x[i].style.backgroundColor = "red";
}
```

Définissez la couleur d'arrière-plan de tous les éléments <p> du document :

```
var x = document.querySelectorAll("p");
var i;
for (i = 0; i < x.length; i++) {
  x[i].style.backgroundColor = "red";
}
```

Définissez la bordure de tous les éléments <a> du document qui ont un attribut "target":

```
var x = document.querySelectorAll("a[target]");
var i;
for (i = 0; i < x.length; i++) {
  x[i].style.border = "10px solid red";
}
```



CHAPITRE 2

CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

- Sélecteurs (simples, multiples...)
- Modes d'Accès aux éléments

02 – CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

Modes d'Accès aux éléments



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()` qui sont des méthodes du mixin `ParentNode` et qu'on va pouvoir implémenter à partir des interfaces `Document` et `Element`.

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

02 – CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

Modes d'Accès aux éléments



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

La méthode getElementById() est une méthode du mixin NonElementParentNode et qu'on va implémenter à partir d'un objet Document. Cette méthode renvoie un objet Element qui représente l'élément dont la valeur de l'attribut id correspond à la valeur spécifiée en argument. La méthode getElementById() 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.

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

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.

En utilisant la méthode getElementsByClassName() avec un objet Document, la recherche des éléments se fera dans tout le document. En l'utilisant avec un objet Element, la recherche se fera dans l'élément en question.

```
//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';
}
```

02 – CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

Modes d'Accès aux éléments



Accéder à un élément en fonction de son identité

La méthode `getElementsByName()` 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).

```
//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';
}
```

02 – CONNAÎTRE LES BASES DE LA MANIPULATION DU DOM EN JAVASCRIPT

Modes d'Accès aux éléments



Accéder directement à des éléments particuliers avec les propriétés de Document

Finalement, l'interface Document fournit également des propriétés qui vont nous permettre d'accéder directement à certains éléments ou qui vont retourner des objets contenant des listes d'éléments.

Les propriétés qui vont le plus nous intéresser ici sont les suivantes :

- La propriété body qui retourne le nœud représentant l'élément body ;
- La propriété head qui retourne le nœud représentant l'élément head ;
- La propriété links qui retourne une liste de tous les éléments a ou area possédant un href avec une valeur ;
- La propriété title qui retourne le titre (le contenu de l'élément title) du document ou permet de le redéfinir ;
- La propriété url qui renvoie l'URL du document sous forme de chaîne de caractères ;
- La propriété scripts qui retourne une liste de tous les éléments script du document ;
- La propriété 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.

```
//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';
```



CHAPITRE 3

MANIPULER LES ÉLÉMENTS HTML

Ce que vous allez apprendre dans ce chapitre :

- Manipulation des éléments (Création, modification, suppression)
- Mise à jour des styles, attributs et classes
- Création DOMMenuObject

 10 heures



CHAPITRE 3

MANIPULER LES ÉLÉMENTS HTML

1. Manipulation des éléments (Création, modification, suppression)
2. Mise à jour des styles, attributs et classes
3. Création DOMMenuObject

03 – MANIPULER LES ÉLÉMENTS HTML

Manipulation des éléments

Créer un élément en JavaScript

Il est courant que les sites Web interactifs créent de nouveaux éléments dynamiquement sur l'action de l'utilisateur et les utilisent.

Alors, comment créer un nouvel élément ?

La façon dont nous créons de nouveaux éléments est d'utiliser la méthode createElement. Appelez la méthode createElement via documentobject et transmettez le nom de la balise de l'élément que vous souhaitez créer.

```
const element = document.createElement('p');
```

La variable element renvoie la référence de l'élément créé.

La méthode createElement convertit le nom de balise de l'élément en minuscules avant de créer l'élément.

```
const element1 = document.createElement('p');
console.log(element1);
const element2 = document.createElement('div');
console.log(element2);
// method Lower case the passed tag name
const element3 = document.createElement('DIV');
console.log(element3);
```

L'élément créé par la méthode createElement() ne s'attache pas automatiquement au document, nous devons explicitement ajouter des éléments au document.

03 – MANIPULER LES ÉLÉMENTS HTML

Manipulation des éléments



Ajouter un élément en JavaScript

Depuis maintenant, nous avons créé l'élément et avons également stocké la référence de l'élément dans une variable, mais l'élément créé flotte sans but car DOM ne connaît pas l'élément créé jusqu'à présent.

Pour que l'élément créé fasse partie du document, nous devons le spécifier comme élément parent et lui ajouter un élément nouvellement créé.

Pour ajouter un élément, utilisez la méthode `append()`. Il insère un ensemble d' objets Node ou d' objets DOMString en tant que dernier enfant du `ParentNode`.

```
<script>
    function addChild() {
        const parent = document.getElementById("parent"); // selecting
        parent
        const child = document.createElement("p"); // creating child
        child.innerHTML = "This is second child"; // adding some content
        parent.append(child);
    }
</script>
```

Supprimer un élément en JavaScript

Dans une application Web interactive, vous souhaiterez non seulement créer et ajouter un nouvel élément au document, mais également supprimer tout élément souhaité.

La méthode `removeChild` supprime un élément de la structure DOM. Le nœud à supprimer est passé en argument à la méthode. La méthode `removeChild` renvoie une référence au nœud enfant qui est supprimé.

```
const parent = document.getElementById("parent"); // selecting parent
const child = document.getElementById("secondChild"); // selecting child

function remove() {
    parent.removeChild(child);
}
```

Modifier un élément en JavaScript

La `replaceChild()` méthode remplace un nœud par un autre nœud dans le DOM.

```
parent.replaceChild(newElement, oldElement)
```

parent - Parent element whose child is to be replaced

newElement - Element which will replace another one

oldElement - Element which will be replaced

The method returns the element which is replaced.

```
const parent = document.getElementById("parent"); // selecting parent
const oldElement = document.getElementById("child"); // selecting oldElement
const newElement = document.createElement("h2"); // creating newElement which is h2

function replace() {
    newElement.innerHTML = "This is new element, old element replaced by this."
    parent.replaceChild(newElement, oldElement);
}
```



CHAPITRE 3

MANIPULER LES ÉLÉMENTS HTML

1. Manipulation des éléments (Création, modification, suppression)
2. Mise à jour des styles, attributs et classes
3. Création DOMMenuObject

03 – MANIPULER LES ÉLÉMENTS HTML

Mise à jour des styles, attributs et classes



Mettre à jour le style

Une fois que nous avons identifié l'élément de manière unique, nous pouvons utiliser les méthodes `.style` ou `.className` pour changer ses styles CSS.

```
<div class="col-md-12">
  <div class="p-3">
    <label>Input String:</label><br>
    <input type="text" class="input-blue-border" id="b1" value="120">
    <button class="m1-3" onclick="changeStyle()">Change Border</button>
  </div>
</div>
```

```
function changeStyle() {
  document.getElementsByClassName("input-blue-border")[0].style.borderColor = "red";
}
```

Notez que dans la méthode `changeStyle()` nous interrogeons l'élément d'entrée avec la méthode `document.getElementsByClassName("input-blue-border")`. Il renvoie un tableau avec des éléments sélectionnés. Nous sélectionnons le premier élément du tableau et changeons sa bordure aquarelle avec `.style.borderColor = "red"`.

03 – MANIPULER LES ÉLÉMENTS HTML

Mise à jour des styles, attributs et classes



Définir le style à l'aide de element.className

On peut utiliser element.className pour changer divers paramètres de style d'un élément HTML en les regroupant comme une classe et en attribuant le nom de la classe à l'élément sélectionné avec element.className. Cette méthode est utile, en particulier dans les scénarios où nous devons afficher une erreur dans un champ de saisie. Dans ce cas, nous devons changer la couleur de la bordure du champ de saisie en couleur rouge et le texte intérieur de l'entrée en couleur rouge. Par conséquent, nous pouvons regrouper ces styles en tant que classe et les affecter à l'élément en utilisant l'attribut element.className. Le code suivant illustre la gestion des erreurs.

```
<div class="col-md-12">
  <div class="p-3">
    <label>Input String:</label><br>
    <input type="text" class="input-blue-border" id="b1" value="120">
    <button class="ml-3" onclick="changeClass()">Change Border</button>
  </div>
</div>
```

```
function changeClass() {
  document.getElementsByClassName("input-blue-border")[0].className = "input-error";
}
```

La classe input-error a quelques attributs qui définissent la couleur de la bordure et la couleur de la police du champ de saisie sur le rouge.

Mise à jour d'un attribut avec setAttribute

La `setAttribute()` méthode est utilisée pour définir un attribut sur l'élément spécifié.

Si l'attribut existe déjà sur l'élément, la valeur est mise à jour ; sinon, un nouvel attribut est ajouté avec le nom et la valeur spécifiés. Le code JavaScript dans l'exemple suivant ajoutera les attributs `class` et `disabled` à l'élément `<button>`.

```
<button type="button" id="myBtn">Click Me</button>
<script>
    // Selecting the element
    var btn = document.getElementById("myBtn");

    // Setting new attributes
    btn.setAttribute("class", "click-btn");
    btn.setAttribute("disabled", "");
</script>
```

Mise à jour d'un attribut avec setAttribute

De même, vous pouvez utiliser la `setAttribute()` méthode pour mettre à jour ou modifier la valeur d'un attribut existant sur un élément HTML. Le code JavaScript dans l'exemple suivant mettra à jour la valeur de l'attribut `href` existant d'un élément `<a>`.

```
<a href="#" id="myLink">Tutorial Republic</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Changing the href attribute value
    link.setAttribute("href", "https://www.tutorialrepublic.com");
</script>
```

Suppression d'attributs d'éléments

La méthode `removeAttribute()` est utilisée pour supprimer un attribut de l'élément spécifié.

Le code JavaScript dans l'exemple suivant supprimera l'attribut `href` d'un élément d'ancre.

```
<a href="https://www.google.com/" id="myLink">Google</a>

<script>
    // Selecting the element
    var link = document.getElementById("myLink");

    // Removing the href attribute
    link.removeAttribute("href");
</script>
```



CHAPITRE 3

MANIPULER LES ÉLÉMENTS HTML

1. Manipulation des éléments (Création, modification, suppression)
2. Mise à jour des styles, attributs et classes
3. Création DOMMenuObject

03 – MANIPULER LES ÉLÉMENTS HTML

Création DOMMenuObject



L'objet de menu DOM HTML en HTML représente l' élément <menu> .

Syntaxe

```
var menuObject = document.createElement("MENU")
```

```
<!DOCTYPE html>
<html>
<head>
<title>Menu Object</title>
<link rel="stylesheet" href="style.css">
<script src="script.js"></script>
</head>
<body>
<form contextmenu="MENU">
<label for="urlSelect">Current URL:</label>
<input type="url" size="30" id="urlSelect" value="https://www.example.com/aboutUs">
<div id="divDisplay"></div>
<menu type="context" id="MENU">
<menuitem Label="Get URL" onclick="contextFunction(1);"></menuitem>
<menuitem Label="Get Hostname" onclick="contextFunction(2);"></menuitem>
</menu>
</form>
</body>
</html>
```

03 – MANIPULER LES ÉLÉMENTS HTML

Création DOMMenuObject



Script.js

```
var divDisplay = document.getElementById("divDisplay");
var urlSelect = document.getElementById("urlSelect");
function gethref(){
    divDisplay.textContent = 'URL Path: '+location.href;
}
function getHostname(){
    divDisplay.textContent = 'Hostname: '+location.hostname;
}
function contextFunction(role){
    if(role === 1)
        this.gethref();
    else
        this.getHostname();
}
```

style.js

```
form {
    width:70%;
    margin: 0 auto;
    text-align: center;
}
* {
    padding: 2px;
    margin:5px;
}
input[type="button"] {
    border-radius: 10px;
}
```

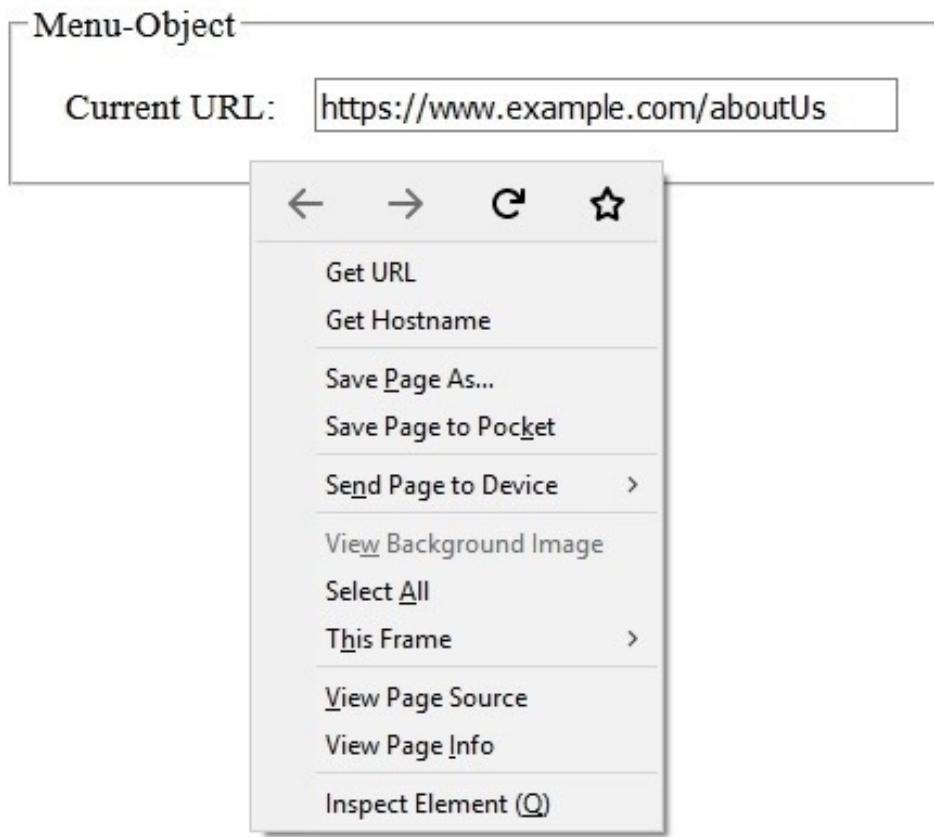
03 – MANIPULER LES ÉLÉMENTS HTML

Création DOMMenuObject

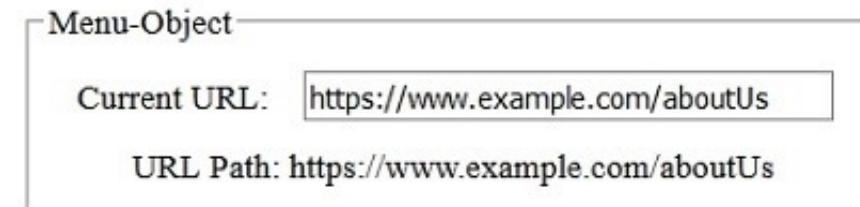


Résultat

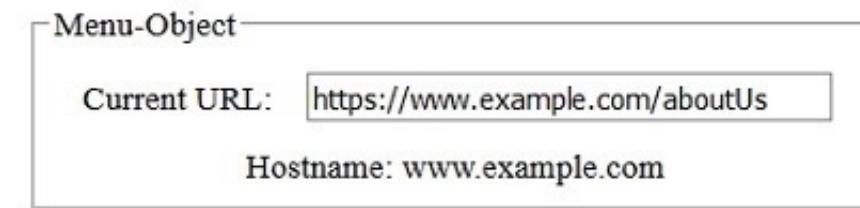
Cela produira la sortie suivante :
Clic droit sur le formulaire



Après avoir cliqué sur l' élément de menu "Obtenir l'URL" dans le menu contextuel



Après avoir cliqué sur l' élément de menu "Obtenir le nom d'hôte" dans le menu contextuel





PARTIE 4

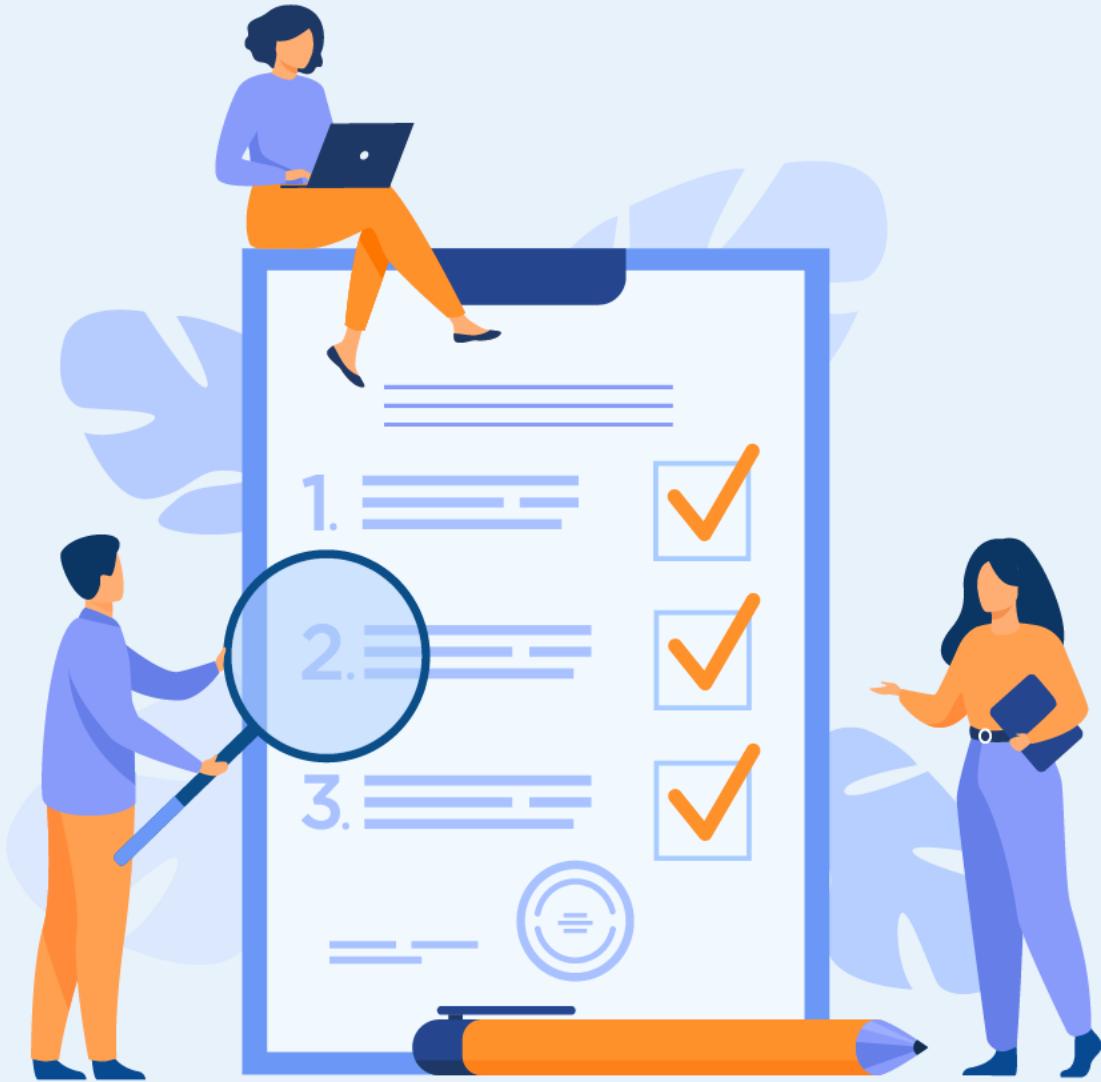
GÉRER LES ÉVÉNEMENTS UTILISATEUR

Dans ce module, vous allez :

- Comprendre la notion d'événement pour gérer l'interactivité
- Gérer les éléments d'un formulaire



 10 heures

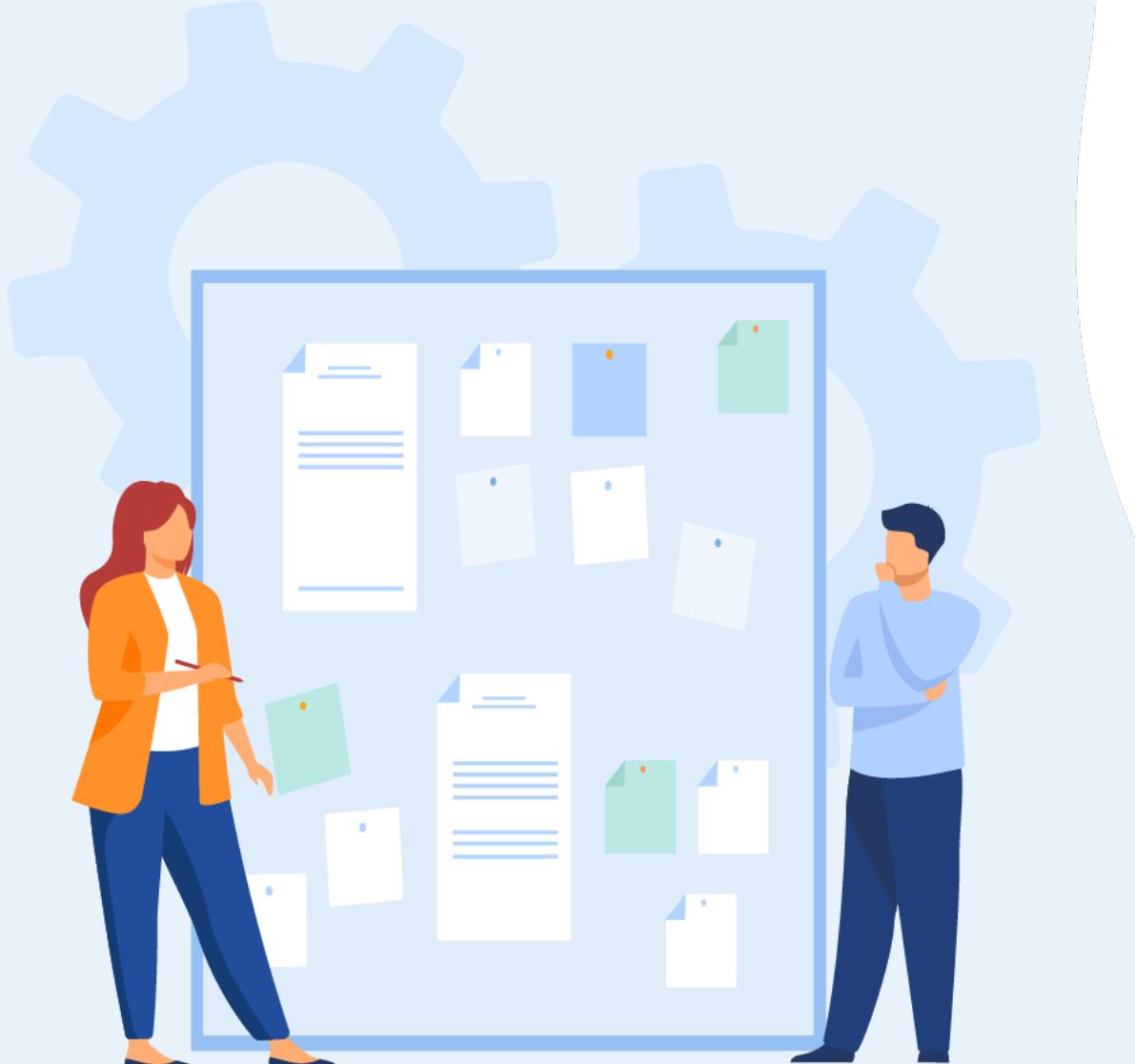


CHAPITRE 1

COMPRENDRE LA NOTION D'ÉVÉNEMENT POUR GÉRER L'INTERACTIVITÉ

Ce que vous allez apprendre dans ce chapitre :

- Définition d'un évènement
- Méthode addEventListener
- MouseEvents
- Interaction avec le clavier



CHAPITRE 1

COMPRENDRE LA NOTION D'ÉVÉNEMENT POUR GÉRER L'INTERACTIVITÉ

1. Définition d'un évènement
2. Méthode addEventListener
3. MouseEvents
4. Interaction avec le clavier

01 – COMPRENDRE LA NOTION D'ÉVÉNEMENT POUR GÉRER L'INTERACTIVITÉ

Définition d'un évènement



Qu'est-ce qu'un événement ?

L'événement est une action ou un état de toute action effectuée soit par l'utilisateur, soit par le navigateur.

Tous les noeuds de DOM génèrent de telles actions, mais les événements ne sont pas limités à DOM, il existe d'autres événements comme un événement de souris, un événement de clavier, un événement de document, etc.

Exemples d'événements DOM :

- L'utilisateur clique sur un bouton
- Souris déplacée
- Page chargée
- Touche enfoncée
- Formulaire soumis

En utilisant Javascript, vous pouvez attacher et réagir à certaines actions. Les événements sont généralement enveloppés d'une fonction, qui s'exécute après que l'événement se soit produit.

En utilisant ces événements, vous pouvez faire tellement de choses car vous avez maintenant le contrôle sur l'action de l'utilisateur. Vous pouvez faire quelque chose lorsque l'utilisateur effectue une action comme cliquer sur un bouton.

01 – COMPRENDRE LA NOTION D'ÉVÉNEMENT POUR GÉRER L'INTERACTIVITÉ

Définition d'un évènement



Terminologie des événements

Il existe deux terminologies lorsque nous rencontrons des événements dans le développement Web :

1. Écouteur d'événement - L'écouteur d'événement est un objet qui écoute un certain événement comme un clic, une pression sur une touche, un mouvement de souris, etc.

2. Gestionnaire d'événements - Les gestionnaires d'événements sont généralement une fonction appelée lorsqu'un certain événement se produit.

Nous pouvons soit attacher directement un événement à l'élément HTML en tant qu'attribut, soit utiliser javascript pour le faire. Voyons en détail.

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

Définition d'un évènement



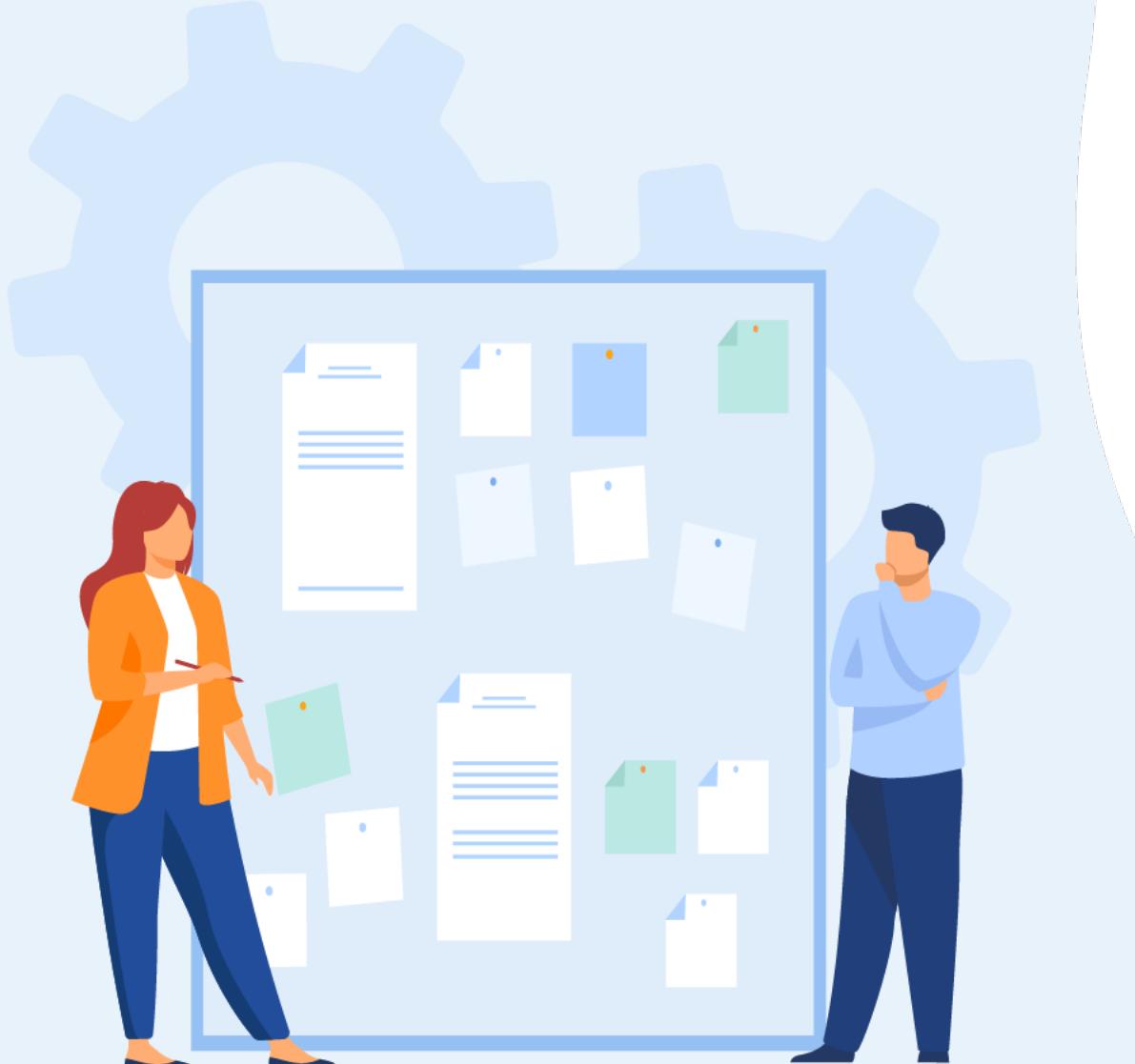
Ajout d'un écouteur d'événement

Les écouteurs d'événement sont quelque chose qui garde une trace de l'action à entreprendre lorsqu'un événement spécifique se produit.

Il écoute l'événement et déclenche la fonction. mais vous devez mentionner spécifiquement dans quel cas quelle action doit être entreprise.

Il existe 3 manières différentes d'attacher un écouteur d'événement à un certain élément :

- 1.Utilisation de l'attribut HTML
- 2.Utilisation de la propriété DOM
- 3.Utilisation de la **méthode addEventListener**



CHAPITRE 1

COMPRENDRE LA NOTION D'ÉVÉNEMENT POUR GÉRER L'INTERACTIVITÉ

1. Définition d'un évènement
2. Méthode `addEventListener`
3. MouseEvents
4. Interaction avec le clavier

01 – COMPRENDRE LA NOTION D'ÉVÉNEMENT POUR GÉRER L'INTERACTIVITÉ

Méthode addEventListener



méthode addEventListener

La méthode **addEventListener** attache un écouteur d'événement à un élément HTML.

Le nouvel événement joint n'affecte pas les événements précédents. Vous pouvez ajouter plusieurs événements à l'élément de la même manière que nous avons utilisé l'attribut HTML.

En utilisant la méthode **addEventListener()**, vous pouvez ajouter des événements à n'importe quel élément DOM, pas nécessairement uniquement des éléments HTML.

Syntaxe addEventListener :

`element.addEventListener(événement, fonction de rappel, useCapture);`

element - C'est un élément HTML auquel nous attachons un événement

événement - C'est le nom de l'événement qui vous intéresse

fonction de rappel - C'est la fonction qui s'exécute après le déclenchement de l'événement

useCapture - Il s'agit d'une valeur booléenne qui spécifie s'il faut utiliser le bouillonnement d'événements ou la capture d'événements. (Ceci est un paramètre facultatif et la valeur par défaut est false)

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

Méthode addEventListener



méthode addEventListener

Exemple

```
const element = document.getElementById("button");

element.addEventListener("click", message);

function message() {
    alert("Button Clicked!")
}
```

Lors de l'utilisation de la méthode **addEventListener()**, n'utilisez pas le préfixe « on » dans le nom de l'événement. comme, au lieu d'utiliser 'onclick', utilisez simplement 'click'.

Vous pouvez également utiliser une fonction interne dans la méthode **addEventListener()**.

```
const element = document.getElementById("button");
element.addEventListener("click", function () { alert("Button Clicked!") });
```

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

Méthode addEventListener



Événement multiple utilisant addEventListener

La méthode **addEventListener** peut très bien ajouter plusieurs méthodes identiques ou différentes à un seul élément. Cela signifie que nous pouvons ajouter 2 ou plus de 2 écouteurs d'événement pour le même événement.

```
const element = document.getElementById("button");
element.addEventListener("click", fxn1);
element.addEventListener("click", fxn2);
function fxn1() {
    alert("Click Event Fired this function");
}
function fxn2() {
    alert("Same click event fired this function too");
}
```

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

Méthode addEventListener



Événement multiple utilisant addEventListener

Plusieurs types d'événements différents peuvent également être attachés au même élément HTML à l'aide de la méthode addEventListener().

```
const element = document.getElementById("button");
element.addEventListener("click", clickFxn);
element.addEventListener("mouseover", mouseoverFxn);
element.addEventListener("mouseout", mouseoutFxn);
function clickFxn() {
    alert("Click Event Fired this function");
}
function mouseoverFxn() {
    element.style.background = "tomato";
    element.style.padding = "8px";
}
function mouseoutFxn() {
    element.style.background = "white";
    element.style.padding = "2px";
}
```

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

Méthode addEventListener



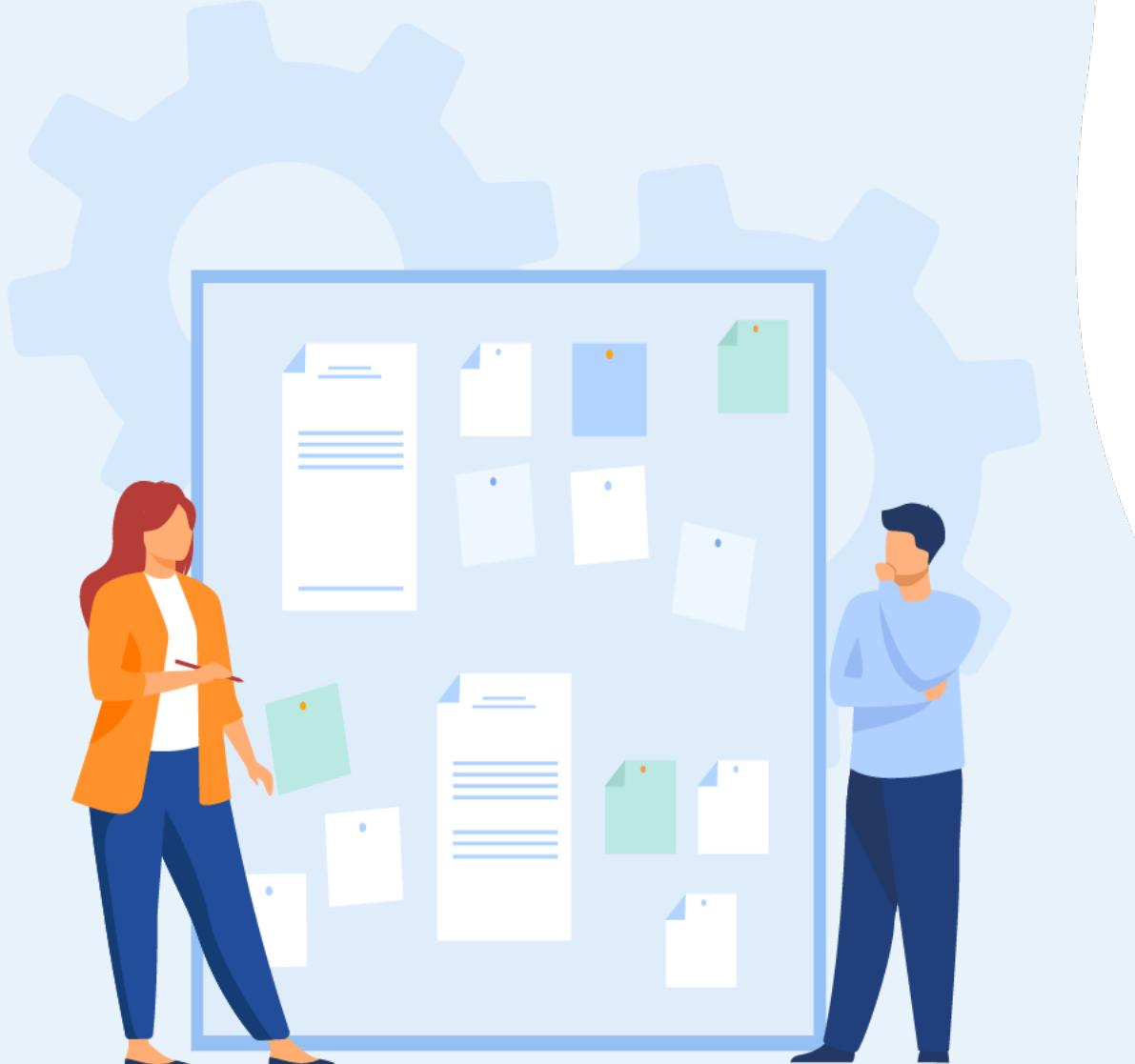
Supprimer l'écouteur d'événement

L'écoute des événements peut également être supprimée de tout élément ou objet HTML.

Pour supprimer l'écouteur d'événement, utilisez la méthode **removeEventListner()**.

```
<p class="tomato">This has "mouseover" event listner.</p>
<button id="button" onclick="removeEvent()">Remove Event listner</button>

<script>
var element = document.querySelector(".tomato");
element.addEventListener("mouseover", createAlert);
function createAlert(){
  alert("Event Triggered!");
}
function removeEvent(){
  element.removeEventListener("mouseover", createAlert);
}
</script>
```



CHAPITRE 1

COMPRENDRE LA NOTION D'ÉVÉNEMENT POUR GÉRER L'INTERACTIVITÉ

1. Définition d'un évènement
2. Méthode addEventListener
3. **MouseEvents**
4. Interaction avec le clavier

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

MouseEvents

Introduction aux événements de souris JavaScript

Les événements de souris se déclenchent lorsque vous utilisez la souris pour interagir avec les éléments de la page. Les événements DOM définissent neuf événements souris.

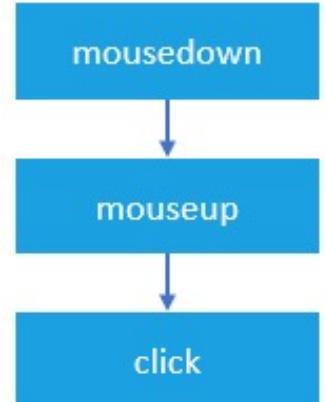
mousedown, mouseup, et click

Lorsque vous cliquez, pas moins de trois événements de souris se déclenchent dans l'ordre suivant :

mousedown se déclenche lorsque vous appuyez sur le bouton de la souris sur l'élément.

mouseup se déclenche lorsque vous relâchez le bouton de la souris sur l'élément.

click se déclenche lorsqu'un **mousedown** et un **mouseup** sont détectés sur l'élément.



Si vous appuyez sur le bouton de la souris sur un élément et déplacez votre souris hors de l'élément, puis relâchez le bouton de la souris. Le seul événement **mousedown** se déclenche sur l'élément.

De même, si vous appuyez sur le bouton de la souris, déplacez la souris sur l'élément et relâchez le bouton de la souris, le seul événement **mouseup** se déclenche sur l'élément.

Dans les deux cas, l'événement **click** ne se déclenche jamais.

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

MouseEvents

Introduction aux événements de souris JavaScript

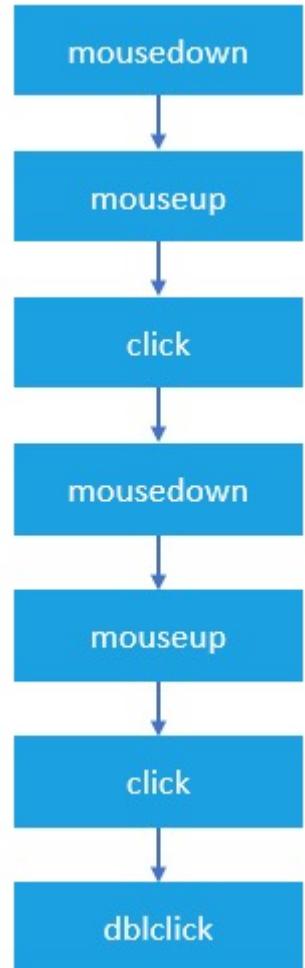
dblclick

En pratique, vous utilisez rarement l'événement dblclick. L'événement dblclick se déclenche lorsque vous double-cliquez sur un élément.

Il faut deux événements de clic pour déclencher un événement dblclick. L'événement dblclick a quatre événements déclenchés dans l'ordre suivant :

1. mousedown
2. mouseup
3. click
4. mousedown
5. mouseup
6. click
7. dblclick

Comme vous pouvez le voir, les événements click ont toujours lieu avant l'événement dblclick. Si vous enregistrez les deux click et les gestionnaires dblclick d'événements sur le même élément, vous ne saurez pas exactement quel utilisateur a réellement cliqué ou double-cliqué sur l'élément.



01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

MouseEvents



Introduction aux événements de souris JavaScript

mousemove

L'événement **mousemove** se déclenche à plusieurs reprises lorsque vous déplacez le curseur de la souris autour d'un élément. Même lorsque vous déplacez la souris d'un pixel, l'événement **mousemove** se déclenche toujours. Cela ralentira la page, par conséquent, vous n'enregistrez le gestionnaire **mousemove** d'événements que lorsque vous en avez besoin et supprimez immédiatement le gestionnaire d'événements dès qu'il n'est plus utilisé, comme ceci :

```
element.onmousemove = mouseMoveEventHandler;  
element.onmousemove = null;
```

mouseover / mouseout

Le **mouseover** se déclenche lorsque le curseur de la souris est à l'extérieur de l'élément, puis se déplace à l'intérieur des limites de l'élément.

Le **mouseout** se déclenche lorsque le curseur de la souris survole un élément, puis déplace un autre élément.

mouseenter / mouseleave

Le **mouseenter** se déclenche lorsque le curseur de la souris est à l'extérieur d'un élément, puis se déplace à l'intérieur des limites de l'élément.

Le **mouseleave** se déclenche lorsque le curseur de la souris survole un élément, puis se déplace vers l'extérieur des limites de l'élément.

Les deux **mouseenter** et **mouseleave** ne font pas de bulles et ne se déclenchent pas lorsque le curseur de la souris se déplace sur les éléments descendants.

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

MouseEvents



Enregistrement des gestionnaires d'événements de souris

Pour enregistrer un événement souris, procédez comme suit :

- Tout d'abord, sélectionnez l'élément à l'aide de la méthode querySelector() ou getElementById().
- Ensuite, enregistrez l'événement de souris à l'aide de la méthode addEventListener().

Par exemple, supposons que vous ayez le bouton suivant :

```
<button id="btn">Click Me!</button>
```

Pour enregistrer un gestionnaire d'événements de clic de souris, vous utilisez le code suivant :

```
let btn = document.querySelector('#btn');

btn.addEventListener('click',(event) => {
    console.log('clicked');
});
```

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

MouseEvents

DéTECTER LES BOUTONS DE LA SOURIS

L'objet event transmis au gestionnaire d'événements de la souris a une propriété appelée button qui indique quel bouton de la souris a été enfoncé pour déclencher l'événement.

Le bouton de la souris est représenté par un nombre :

- 0 : le bouton principal de la souris enfoncé, généralement le bouton gauche.
- 1 : le bouton auxiliaire enfoncé, généralement le bouton du milieu ou le bouton de la roue.
- 2 : le bouton secondaire enfoncé, généralement le bouton droit.
- 3 : le quatrième bouton enfoncé, généralement le bouton Précédent du navigateur .
- 4 : le cinquième bouton enfoncé, généralement le bouton Suivant du navigateur .

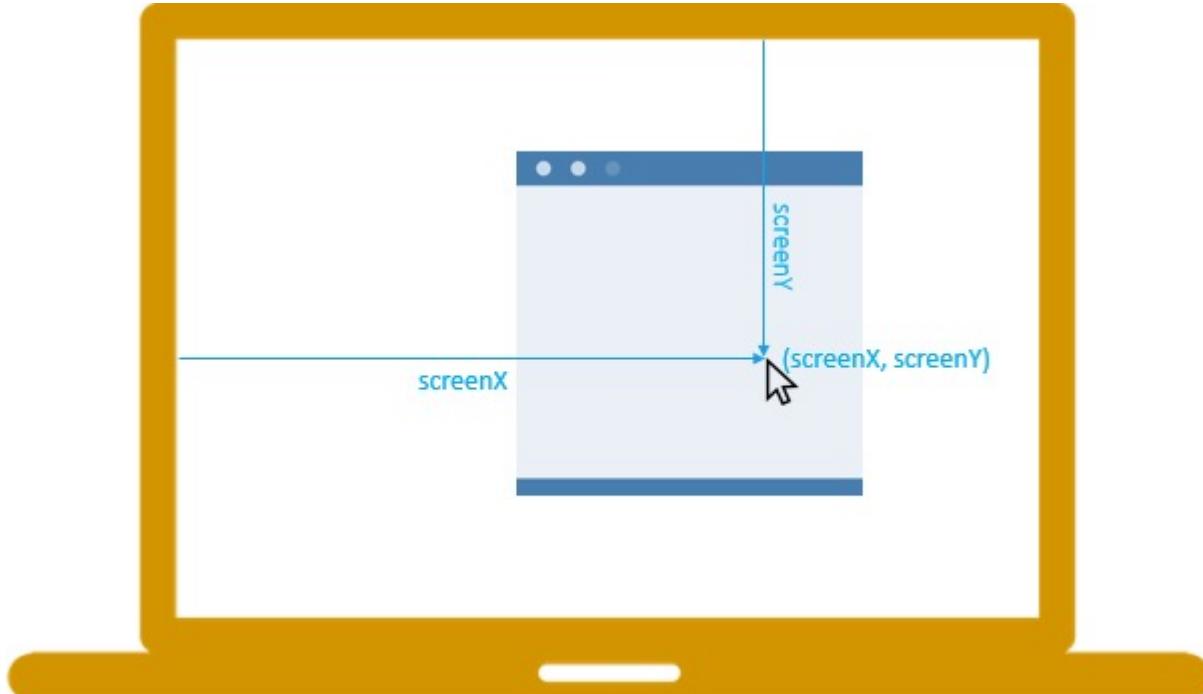


01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

MouseEvents

Obtenir les coordonnées de l'écran

Les propriétés screenX et screenY de l'événement passées au gestionnaire d'événements de la souris renvoient les coordonnées d'écran de l'emplacement de la souris par rapport à l'ensemble de l'écran.

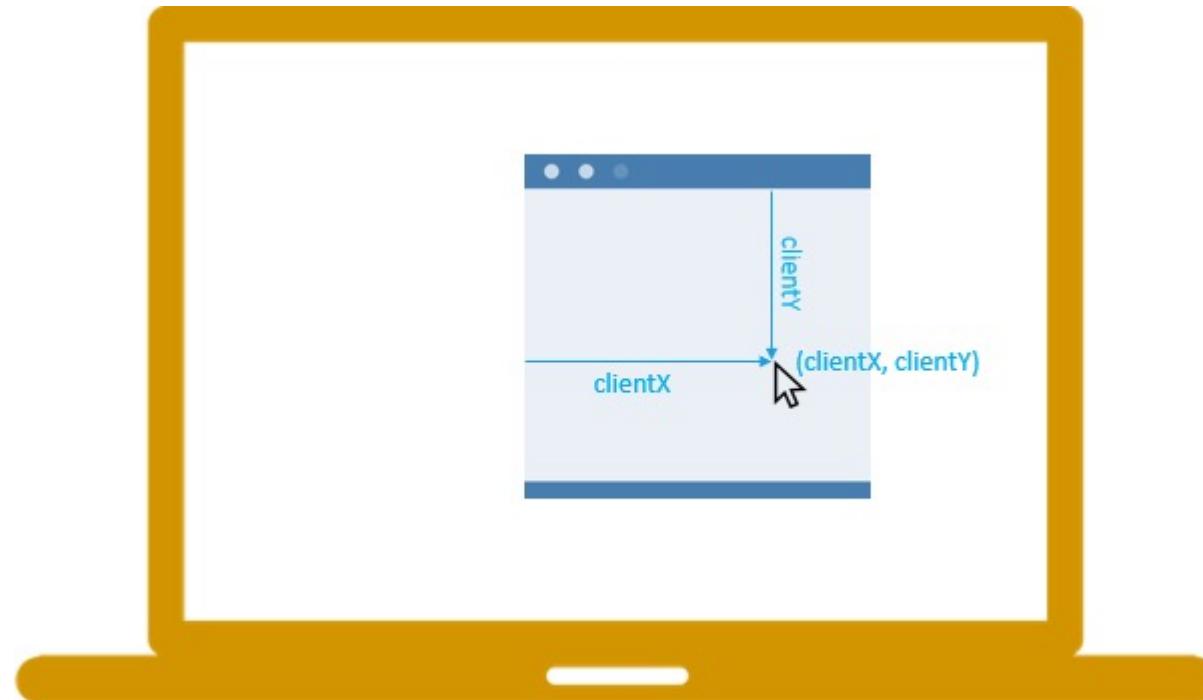


01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

MouseEvents

Obtenir les coordonnées de l'écran

D'autre part, les propriétés `clientX` et `clientY` fournissent les coordonnées horizontales et verticales dans la zone cliente de l'application où l'événement de souris s'est produit :



01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

MouseEvents



Exemple

```
<!DOCTYPE html>
<html>
<head>
    <title>JS Mouse Location Demo</title>
    <style>
        #track { background-color: goldenrod; height: 200px; width: 400px; }
    </style>
</head>
<body>
    <p>Faire bouger la souris pour voir sa position.</p>
    <div id="track"></div>
    <p id="log"></p>
    <script>
        let track = document.querySelector('#track');
        track.addEventListener('mousemove', (e) => {
            let log = document.querySelector('#log');
            log.innerText = `Screen X/Y: (${e.screenX}, ${e.screenY}) Client X/Y: (${e.clientX}, ${e.clientY})`;
        });
    </script>
</body>
</html>
```



CHAPITRE 1

COMPRENDRE LA NOTION D'ÉVÉNEMENT POUR GÉRER L'INTERACTIVITÉ

1. Définition d'un évènement
2. Méthode addEventListener
3. MouseEvents
4. Interaction avec le clavier

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

Interaction avec le clavier



Introduction aux événements de clavier JavaScript

Lorsque vous interagissez avec le clavier, les événements de clavier sont déclenchés. Il existe trois principaux événements de clavier :

keydown – se déclenche lorsque vous appuyez sur une touche du clavier et il se déclenche à plusieurs reprises pendant que vous maintenez la touche enfoncée.

keyup – se déclenche lorsque vous relâchez une touche du clavier.

keypress – se déclenche lorsque vous appuyez sur un clavier de caractères comme a, b, ou c, pas sur la touche fléchée gauche, le clavier d'accueil ou de fin, ... Le **keypress** se déclenche également à plusieurs reprises lorsque vous maintenez la touche du clavier enfoncée.

Les événements de clavier se déclenchent généralement sur la zone de texte, car tous les éléments les prennent en charge.

Lorsque vous appuyez une fois sur une touche de caractère du clavier, trois événements de clavier sont déclenchés dans l'ordre suivant :

- **keydown**
- **keypress**
- **Keyup**

Les événements **keydown** et **keypress** sont déclenchés avant toute modification apportée à la zone de texte, tandis que l'événement **keyup** se déclenche après que les modifications ont été apportées à la zone de texte. Si vous maintenez une touche de caractère enfoncée, les touches **keydown** et **keypress** sont déclenchées à plusieurs reprises jusqu'à ce que vous relâchiez la touche.

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

Interaction avec le clavier



Gestion des événements clavier

Pour gérer un événement clavier, procédez comme suit :

Tout d'abord, sélectionnez l'élément sur lequel l'événement clavier se déclenchera. Il s'agit généralement d'une zone de texte.

Ensuite, utilisez l'élément **addEventListener()** pour enregistrer un gestionnaire d'événements.

Supposons que vous ayez la zone de texte suivante avec l'identifiant message :

```
<input type="text" id="message">
```

Ce qui suit illustre comment enregistrer des écouteurs d'événement de clavier :

```
let msg = document.getElementById('#message');
msg.addEventListener("keydown", (event) => {
    // handle keydown
});

msg.addEventListener("keypress", (event) => {
    // handle keypress
});

msg.addEventListener("keyup", (event) => {
    // handle keyup
});
```

Si vous appuyez sur une touche de caractère, les trois gestionnaires d'événements seront appelés.

01 – COMPRENDRE LA NOTION D’ÉVÉNEMENT POUR GÉRER L’INTERACTIVITÉ

Interaction avec le clavier



Les propriétés de l’événement clavier

L’événement clavier a deux propriétés importantes : key et code. La propriété key renvoie le caractère qui a été enfoncé alors que la propriété code renvoie le code de touche physique.

Par exemple, si vous appuyez sur la touche z caractère, les event.key retourne z et les event.code retourne KeyZ.

Voir l'exemple suivant :

Si vous tapez caractère z, vous verrez le message suivant :

key=z,code=KeyZ

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Keyboard Events: Key/Code</title>
</head>
<body>
    <input type="text" id="message">

    <script>
        let textBox = document.getElementById('message');
        textBox.addEventListener('keydown', (event) => {
            console.log(`key=${event.key},code=${event.code}`);
        });
    </script>
</body>
</html>
```



CHAPITRE 2

GÉRER LES ÉLÉMENTS D'UN FORMULAIRE

Ce que vous allez apprendre dans ce chapitre :

- Soumission d'un formulaire
- Interruption d'un formulaire
- Validation d'un formulaire



CHAPITRE 1

GÉRER LES ÉLÉMENTS D'UN FORMULAIRE

1. Soumission d'un formulaire
2. Interruption d'un formulaire
3. Validation d'un formulaire

02 – GÉRER LES ÉLÉMENS D'UN FORMULAIRE

Soumission d'un formulaire



La méthode HTML DOM Form submit() est utilisée pour soumettre les données du formulaire à l'adresse spécifiée par l'attribut action. Il agit comme un bouton de soumission pour soumettre les données du formulaire et ne prend aucun type de paramètres.

Syntaxe

Voici la syntaxe de la méthode Form submit()

```
formObject.submit()
```

Exemple

```
<body>
<h1>Form reset() method example</h1>
<form id="FORM1" method="post" action="/sample_page.php">
<label>User Name <input type="text" name="usrN"></label><br>
<label>Age <input type="text" name="Age"><label> <br>
<input type="submit" onclick="SubmitForm()" value="SUBMIT">
<input type="button" onclick="ResetForm()" value="RESET">
</form>
<p id="Sample"></p>
</body>
```

```
function ResetForm() {
    document.getElementById("FORM1").reset();
    document.getElementById("Sample").innerHTML
    = "Form has been reset";
}
```

```
function SubmitForm() {
    document.getElementById("FORM1").submit();
}
```



CHAPITRE 1

GÉRER LES ÉLÉMENTS D'UN FORMULAIRE

1. Soumission d'un formulaire
2. Interruption d'un formulaire
3. Validation d'un formulaire

Empêcher la soumission d'un formulaire

La fonction `preventDefault()` ne laisse pas se dérouler l'action par défaut de l'événement. Le `window.history.back()` nous renvoie à l'URL précédente de notre historique. Toutes les méthodes discutées incluront ces deux fonctions.

```
<form onsubmit="submitForm(event)">
    <input type="text">
    <button type="submit">Submit</button>
</form>
<script type="text/JavaScript">
    function submitForm(event){
        event.preventDefault();
        window.history.back();
    }
</script>
```



CHAPITRE 1

GÉRER LES ÉLÉMENTS D'UN FORMULAIRE

1. Soumission d'un formulaire
2. Interruption d'un formulaire
3. Validation d'un formulaire

02 – GÉRER LES ÉLÉMENS D'UN FORMULAIRE

Validation d'un formulaire



Les propriétés de l'événement clavier

La validation du formulaire HTML peut être effectuée par JavaScript.

Si un champ de formulaire (fname) est vide, cette fonction alerte un message et renvoie false pour empêcher la soumission du formulaire :

```
function validateForm() {
    let x = document.forms["myForm"]["fname"].value;
    if (x == "") {
        alert("Name must be filled out");
        return false;
    }
}
```

```
<form name="myForm" action="/action_page.php" onsubmit="return validateForm()" method="post">
    Name: <input type="text" name="fname">
    <input type="submit" value="Submit">
</form>
```

La fonction peut être appelée lors de la soumission du formulaire :

02 – GÉRER LES ÉLÉMENS D'UN FORMULAIRE

Validation d'un formulaire



Validation automatique des formulaires HTML

La validation du formulaire HTML peut être effectuée automatiquement par le navigateur :

Si un champ de formulaire (fname) est vide, l'attribut required empêche la soumission de ce formulaire :

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
```

02 – GÉRER LES ÉLÉMENTS D'UN FORMULAIRE

Validation d'un formulaire



La validation des données

La validation des données est le processus permettant de s'assurer que l'entrée de l'utilisateur est propre, correcte et utile.

Les tâches de validation typiques sont :

- L'utilisateur a-t-il rempli tous les champs obligatoires ?
- L'utilisateur a-t-il entré une date valide ?
- L'utilisateur a-t-il saisi du texte dans un champ numérique ?
- Le plus souvent, le but de la validation des données est d'assurer une entrée correcte de l'utilisateur.

La validation peut être définie par de nombreuses méthodes différentes et déployée de différentes manières.

La validation côté serveur est effectuée par un serveur Web, une fois que l'entrée a été envoyée au serveur.

La validation côté client est effectuée par un navigateur Web, avant que l'entrée ne soit envoyée à un serveur Web.

02 – GÉRER LES ÉLÉMENS D'UN FORMULAIRE

Validation d'un formulaire



Validation des contraintes HTML

HTML5 a introduit un nouveau concept de validation HTML appelé validation de contrainte .

La validation des contraintes HTML est basée sur :

- Validation des contraintes Attributs d'entrée HTML
- Validation des contraintes CSS Pseudo sélecteurs
- Validation des contraintes Propriétés et méthodes du DOM

Attributs d'entrée HTML de validation de contrainte

Attribut	Description
disabled	L'input doit être désactivé
max	Spécifier la valeur maximale d'un élément input
min	Spécifier la valeur minimale d'un élément input
pattern	Spécifier un modèle de chaîne (Regex)
required	Saisie obligatoire
type	Spécifier le type d'un élément input

02 – GÉRER LES ÉLÉMENTS D'UN FORMULAIRE

Validation d'un formulaire



Validation des contraintes HTML

Validation des contraintes CSS pseudo-sélecteurs

Sélecteur	Description
:disabled	Sélectionner les éléments désactivés
:invalid	Sélectionner les éléments dont la valeur est invalide
:optional	Sélectionne les éléments d'entrée sans attribut "requis" spécifié
:required	Sélectionne les éléments d'entrée avec l'attribut "requis" spécifié
:valid	Sélectionne les éléments d'entrée avec des valeurs valides



PARTIE 5

MANIPULER JQUERY

Dans ce module, vous allez :

- Découvrir jQuery
- Découvrir AJAX



18 heures



CHAPITRE 1

DÉCOUVRIR JQUERY

Ce que vous allez apprendre dans ce chapitre :

- Fonctions essentielles et chaînage
- Comportement des liens
- Association d'évènements et déclenchement
- Intégration de plugins existants
- Utilisation de plugins existants



08 heures



CHAPITRE 1

DÉCOUVRIR JQUERY

1. Fonctions essentielles et chaînage
2. Comportement des liens
3. Association d'évènements et déclenchement
4. Intégration de plugins existants
5. Utilisation de plugins existants

Qu'est-ce que jQuery ?

jQuery est une bibliothèque JavaScript légère, "écrivez moins, faites plus".

Le but de jQuery est de rendre beaucoup plus facile l'utilisation de JavaScript sur votre site Web.

jQuery prend beaucoup de tâches courantes qui nécessitent de nombreuses lignes de code JavaScript à accomplir et les encapsule dans des méthodes que vous pouvez appeler avec une seule ligne de code.

jQuery simplifie également beaucoup de choses compliquées de JavaScript, comme les appels AJAX et la manipulation DOM.

La bibliothèque jQuery contient les fonctionnalités suivantes :

- Manipulation HTML/DOM
- Manipulation CSS
- Méthodes d'événement HTML
- Effets et animations
- AJAX
- Utilitaires

De plus, jQuery propose des plugins pour presque toutes les tâches.

Téléchargement de jQuery

Il existe deux versions de jQuery disponibles au téléchargement :

Version de production - c'est pour votre site Web en direct car il a été minifié et compressé

Version de développement - c'est pour les tests et le développement (code non compressé et lisible)

Les deux versions peuvent être téléchargées sur [jQuery.com](https://jquery.com) .

La bibliothèque jQuery est un fichier JavaScript unique, et vous le référez avec la `<script>` balise HTML (notez que la `<script>` balise doit être à l'intérieur de la section `<head>`) :

```
<head>
  <script src="jquery-3.5.1.min.js"></script>
</head>
```

CDN jQuery

Si vous ne souhaitez pas télécharger et héberger jQuery vous-même, vous pouvez l'inclure à partir d'un CDN (Content Delivery Network).

Google est un exemple de quelqu'un qui héberge jQuery :

```
<head>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
</head>
```

Syntaxe jQuery

La syntaxe jQuery est conçue sur mesure pour sélectionner des éléments HTML et effectuer certaines actions sur le ou les éléments.

La syntaxe de base est : `$(selector).action ()`

Un signe \$ pour définir/accéder à jQuery

Un (sélecteur) pour "interroger (ou rechercher)" des éléments HTML

Une action jQuery () à effectuer sur le(s) élément(s)

Exemples:

`$(this).hide()` - masque l'élément courant.

`$("p").hide()` - masque tous les éléments <p>.

`$(".test").hide()` - masque tous les éléments avec class="test".

`$("#test").hide()` - masque l'élément avec id="test".

L'événement ready pour le document

Vous avez peut-être remarqué que toutes les méthodes jQuery de nos exemples se trouvent dans un événement prêt pour le document :

```
$('document').ready(function(){  
    // jQuery methods go here...  
});
```

Cela permet d'empêcher l'exécution de tout code jQuery avant la fin du chargement du document (est prêt).

Il est recommandé d'attendre que le document soit complètement chargé et prêt avant de travailler dessus. Cela vous permet également d'avoir votre code JavaScript avant le corps de votre document, dans la section head.

Voici quelques exemples d'actions qui peuvent échouer si les méthodes sont exécutées avant que le document ne soit complètement chargé :

- Essayer de masquer un élément qui n'est pas encore créé
- Essayer d'obtenir la taille d'une image qui n'est pas encore chargée

01 – DÉCOUVRIR JQUERY

Fonctions essentielles et chaînage



L'événement ready pour le document

Jusqu'à présent, nous avons écrit des instructions jQuery une par une (l'une après l'autre).

Cependant, il existe une technique appelée chaînage, qui nous permet d'exécuter plusieurs commandes jQuery, l'une après l'autre, sur le(s) même(s) élément(s).

Astuce : De cette façon, les navigateurs n'ont pas à trouver le ou les mêmes éléments plusieurs fois.

Pour enchaîner une action, il suffit d'ajouter l'action à l'action précédente.

Les chaînes exemple suivant ainsi les css(), slideUp() et les méthodes slideDown(). L'élément "p1" devient d'abord rouge, puis il glisse vers le haut, puis vers le bas :

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
```

Nous aurions également pu ajouter plus d'appels de méthode si nécessaire.

Astuce : Lors du chaînage, la ligne de code peut devenir assez longue. Cependant, jQuery n'est pas très strict sur la syntaxe ; vous pouvez la formater comme vous le souhaitez, y compris les sauts de ligne et les retraits.

Cela fonctionne aussi très bien :

```
$("#p1").css("color", "red")
  .slideUp(2000)
  .slideDown(2000);
```

01 – DÉCOUVRIR JQUERY

Fonctions essentielles et chaînage



L'événement ready pour le document

Jusqu'à présent, nous avons écrit des instructions jQuery une par une (l'une après l'autre).

Cependant, il existe une technique appelée chaînage, qui nous permet d'exécuter plusieurs commandes jQuery, l'une après l'autre, sur le(s) même(s) élément(s).

Astuce : De cette façon, les navigateurs n'ont pas à trouver le ou les mêmes éléments plusieurs fois.

Pour enchaîner une action, il suffit d'ajouter l'action à l'action précédente.

Les chaînes exemple suivant ainsi les css(), slideUp() et les méthodes slideDown(). L'élément "p1" devient d'abord rouge, puis il glisse vers le haut, puis vers le bas :

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
```

Nous aurions également pu ajouter plus d'appels de méthode si nécessaire.

Astuce : Lors du chaînage, la ligne de code peut devenir assez longue. Cependant, jQuery n'est pas très strict sur la syntaxe ; vous pouvez la formater comme vous le souhaitez, y compris les sauts de ligne et les retraits.

Cela fonctionne aussi très bien :

```
$("#p1").css("color", "red")
  .slideUp(2000)
  .slideDown(2000);
```



CHAPITRE 1

DÉCOUVRIR JQUERY

1. Fonctions essentielles et chaînage
2. Comportement des liens
3. Association d'évènements et déclenchement
4. Intégration de plugins existants
5. Utilisation de plugins existants

01 – DÉCOUVRIR JQUERY

Comportement des liens

Comment désactiver un lien href en JQuery

Le comportement cliquable du lien « ofppt » dans l'exemple suivant est désactivé en supprimant l'attribut « href » du lien à l'aide de la méthode removeAttr() de jQuery.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>jQuery Removing Clickable Behavior</title>
    <script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
    <script type="text/javascript">
      $(document).ready(function(){
        $(".menu a").each(function(){
          if($(this).hasClass("disabled")){
            $(this).removeAttr("href");
          }
        });
      });
    </script>
  </head>
```

```
<body>
  <ul class="menu">
    <li><a href="https://waytolearnx.com/">Astuces</a></li>
    <li><a href="https://waytolearnx.com/">QCM</a></li>
    <li><a href="https://waytolearnx.com/">Forum</a></li>
    <li><a href="https://waytolearnx.com/" class="disabled">Tutoriel</a></li>
  </ul>
</body>
</html>
```



CHAPITRE 1

DÉCOUVRIR JQUERY

1. Fonctions essentielles et chaînage
2. Comportement des liens
3. **Association d'évènements et déclenchement**
4. Intégration de plugins existants
5. Utilisation de plugins existants

01 – DÉCOUVRIR JQUERY

Association d'événements et déclenchement

Dans jQuery, la plupart des événements DOM ont une méthode jQuery équivalente.

Pour attribuer un événement de clic à tous les paragraphes d'une page, vous pouvez procéder comme suit :

```
$("p").click();
```

L'étape suivante consiste à définir ce qui doit se passer lorsque l'événement se déclenche. Vous devez passer une fonction à l'événement :

```
$("p").click(function(){
    // action goes here!!
});
```

Méthodes d'événement jQuery couramment utilisées

`$(document).ready()`

La méthode `$(document).ready()` nous permet d'exécuter une fonction lorsque le document est complètement chargé.

`click()`

La méthode `click()` attache une fonction de gestionnaire d'événements à un élément HTML.

La fonction est exécutée lorsque l'utilisateur clique sur l'élément HTML.

L'exemple suivant indique : Lorsqu'un événement click se déclenche sur un `<p>` élément ; masquer l' `<p>` élément actuel :

```
$("p").click(function(){
    $(this).hide();
});
```

`dblclick()`

La méthode `dblclick()` attache une fonction de gestionnaire d'événements à un élément HTML.

La fonction est exécutée lorsque l'utilisateur double-clique sur l'élément HTML :

```
$("p").dblclick(function(){
    $(this).hide();
});
```

Méthodes d'événement jQuery couramment utilisées

mouseenter()

La méthode `mouseenter()` attache une fonction de gestionnaire d'événements à un élément HTML.

La fonction est exécutée lorsque le pointeur de la souris entre dans l'élément HTML :

```
$("#p1").mouseenter(function(){
    alert("You entered p1!");
});
```

mouseleave()

La méthode `mouseleave()` attache une fonction de gestionnaire d'événements à un élément HTML.

La fonction est exécutée lorsque le pointeur de la souris quitte l'élément HTML ::

```
$("#p1").mouseleave(function(){
    alert("Bye! You now leave p1!");
});
```



CHAPITRE 1

DÉCOUVRIR JQUERY

1. Fonctions essentielles et chaînage
2. Comportement des liens
3. Association d'évènements et déclenchement
4. **Intégration de plugins existants**
5. Utilisation de plugins existants

Où trouver un plugin ?

La bibliothèque jQuery a été écrite de telle sorte qu'il est très simple de l'étendre en installant des modules additionnels, connus sous le nom d'extensions ou de plugins. De nombreux sites Web se sont spécialisés dans les plugins jQuery.

Pour vous faire gagner du temps, je vais limiter (du moins dans un premier temps) vos recherches à deux sites : The Ultimate jQuery List et jQuery Plugins. Tous deux très bien faits, ils donnent accès à de très nombreux plugins classés par catégories.

Il vous suffit donc d'aller dans une catégorie et de regarder les plugins proposés. Sur The Ultimate jQuery List, cliquez sur un plugin pour en avoir une description. Si le plugin vous intéresse, rendez-vous sur le site Web dédié afin de le télécharger ; la plupart du temps, la documentation du plugin s'y trouve également. Vous trouverez également souvent une démonstration, ce qui est toujours intéressant pour se décider.

Vous allez voir qu'utiliser un plugin est la plupart du temps un jeu d'enfant. Nous allons utiliser le plugin « Websanova Color Picker », proposé sur le site The Ultimate jQuery List. Rendez-vous donc sur ce site, allez dans la catégorie « Color Pickers » et cliquez sur Websanova Color Picker, puis sur Visit Website. Une fois sur le site en question, téléchargez le plugin (il s'agit d'un fichier compressé, choisissez donc le format qui vous convient).

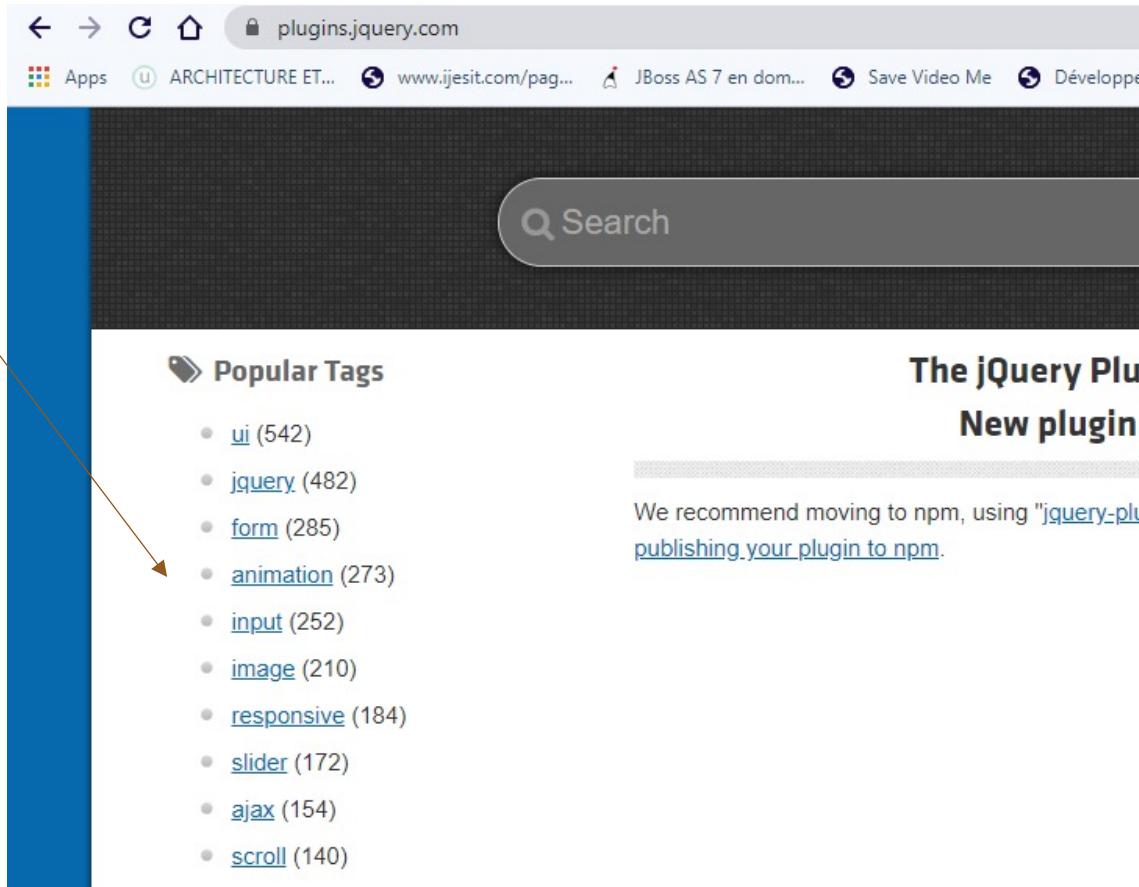
Décompressez l'archive et copiez la version minimisée des fichiers JavaScript et CSS dans le dossier dans lequel vous faites vos développements jQuery. Vous êtes maintenant prêts à utiliser le plugin. Il ne vous reste plus qu'à consulter la documentation. Dans notre cas, elle se trouve en ligne. La figure suivante vous montre à quoi elle ressemble.

Où trouver un plugin ?

Vous pouvez chercher sur internet sur les plugin jQuery. Surement vous allez tomber sur le site web <https://plugins.jquery.com/>. Il existe d'autres sites qui offrent des plugins. Tous ces sites sont utilisés de la même manière.

Intégrer un plugin

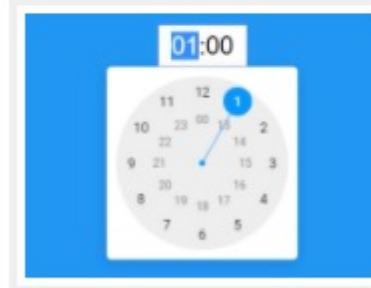
Vous trouvez une liste de catégories où se trouvent plusieurs plugins



Intégrer un plugin

Si vous écrivez le mot clé "time" dans la barre de recherche du site, vous allez trouver un exemple de plugins comme ceci

Free jQuery Plugins About 'time'



[Demo](#) [Download](#)

Android-style Analog Clock time Picker Plugin With jQuery
about a month ago - Time & Clock - 10991 Views
A fancy jQuery **time**picker plugin which lets you create an interactive, Android inspired, Analog Clock-style **time** picker for your input field.

Vous pouvez voir une démonstration avant de le télécharger



CHAPITRE 1

DÉCOUVRIR JQUERY

1. Fonctions essentielles et chaînage
2. Comportement des liens
3. Association d'évènements et déclenchement
4. Intégration de plugins existants
5. Utilisation de plugins existants

Une fois vous cliquez sur "Télécharger", vous allez trouver un tutoriel d'utilisation comme suit:

How to use it:

1. Create a normal text field and set the initial time in the `value` attribute.

```
1 | <input class="time" type="text" value="14:30" />
```

2. Download the plugin, and then insert the JavaScript file

`jquery-clock-timepicker.min.js` after jQuery but before the closing body tag.

```
1 | <script src="//code.jquery.com/jquery.min.js"></script>
2 | <script src="jquery-clock-timepicker.min.js"></script>
```

3. Call the function `clockTimePicker` on the input field and done.

```
1 | $('.time').clockTimePicker();
```

4. All default settings which can be overridden with the options argument.

```
01 | $('.time').clockTimePicker({
02 |
03 | })
```

Chaque plugin a sa propre méthode d'utilisation. Il faut juste suivre le tutoriel fourni par le développeur du plugin



CHAPITRE 2

DÉCOUVRIR AJAX

Ce que vous allez apprendre dans ce chapitre :

- Introduction à AJAX
- Fonctionnement d'AJAX
- Implémentation d'AJAX via jQuery

 10 heures



CHAPITRE 2

DÉCOUVRIR AJAX

1. Introduction à AJAX
2. Fonctionnement d'AJAX
3. Implémentation d'AJAX via jQuery

01 – DÉCOUVRIR AJAX

Introduction à AJAX

Qu'est-ce qu'AJAX ?

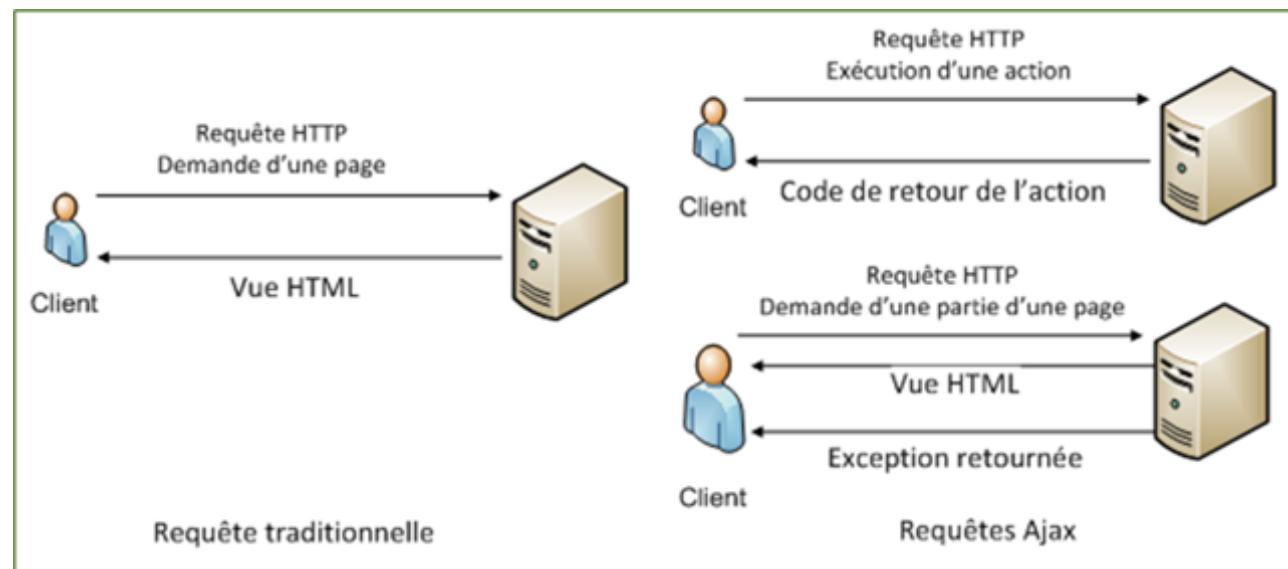
AJAX = Asynchronous JavaScript And XML.

AJAX n'est pas un langage de programmation.

AJAX utilise simplement une combinaison de :

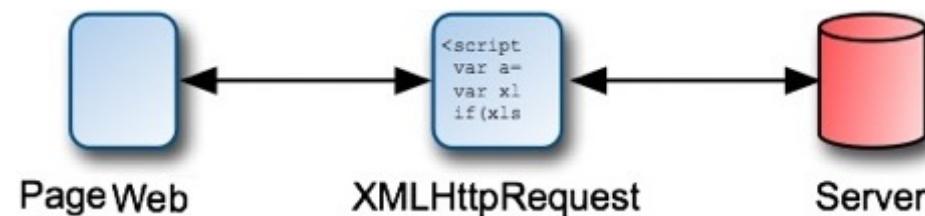
- Un objet XMLHttpRequest intégré au navigateur (pour demander des données à un serveur Web)
- JavaScript et HTML DOM (pour afficher ou utiliser les données)

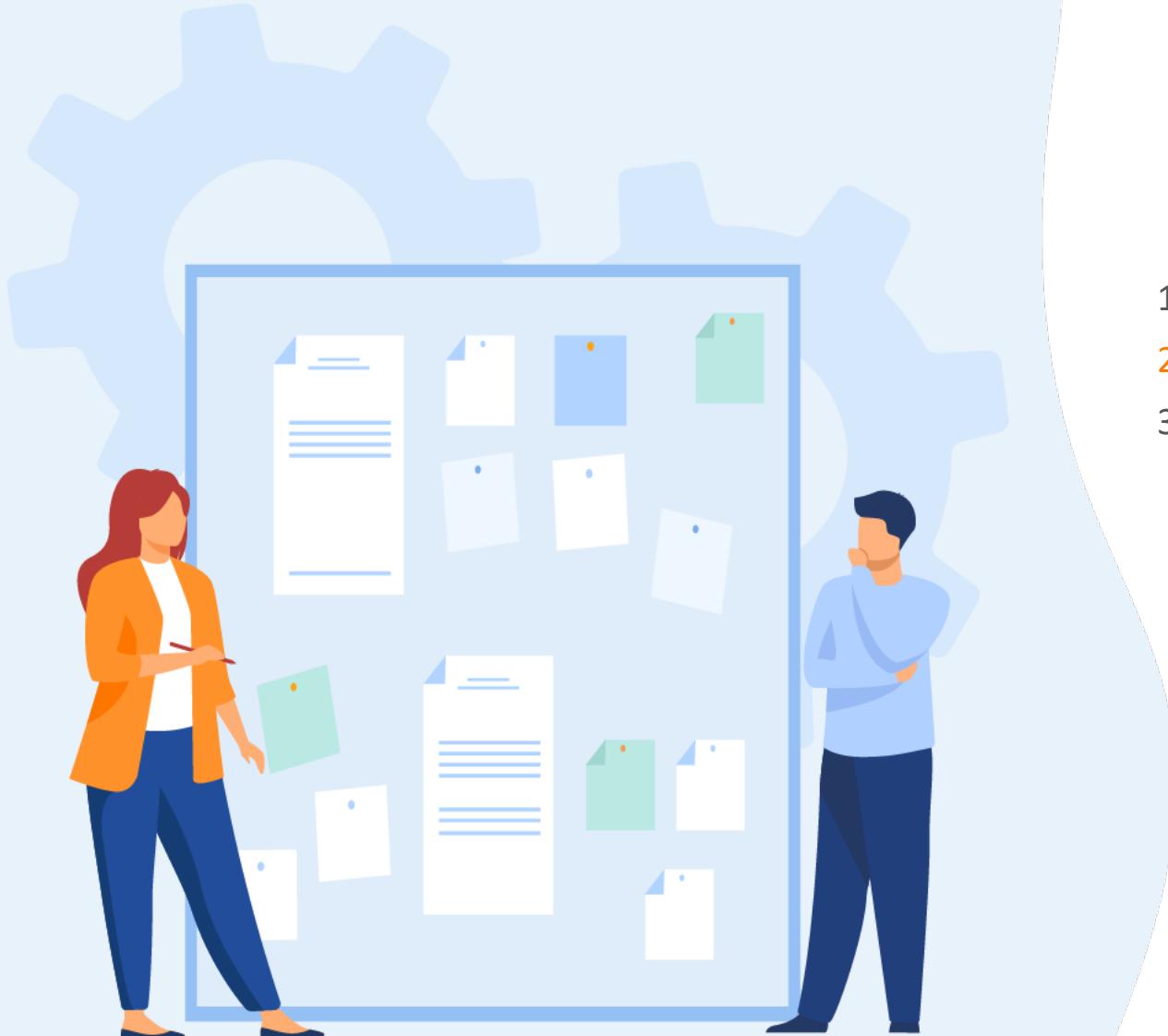
AJAX permet aux pages Web d'être mises à jour de manière asynchrone en échangeant des données avec un serveur Web dans les coulisses. Cela signifie qu'il est possible de mettre à jour des parties d'une page Web, sans recharger la page entière.



L'objet XMLHttpRequest

- L'objet XMLHttpRequest permet de faire des requêtes HTTP au serveur, de recevoir des réponses et de mettre à jour une partie de la page Web.
- En mode asynchrone, cette mise à jour se réalise sans devoir recharger la page et donc de façon totalement transparente pour l'utilisateur.
- XMLHttpRequest est l'objet qui va vous permettre d'utiliser la technologie Ajax au sein de vos pages web.
- XMLHttpRequest est basé sur le principe d'échange de données entre un client (la page WEB de l'internaute qui prend l'initiative de la connexion) et un serveur sur lequel se trouve la page ou l'application à laquelle la page Web veut accéder.
- L'objet XMLHttpRequest s'utilise donc dans une architecture de type client-serveur.





CHAPITRE 2

DÉCOUVRIR AJAX

1. Introduction à AJAX
2. Fonctionnement d'AJAX
3. Implémentation d'AJAX via jQuery

02 – DÉCOUVRIR AJAX

Fonctionnement d'AJAX

Comment fonctionne AJAX?

L'objet XMLHttpRequest a été créé pour permettre aux développeurs d'initier des demandes HTTP à partir de n'importe où dans une application.

L'objet XMLHttpRequest est créé par le moteur JavaScript du navigateur.

Cet objet est alors utilisé pour envoyer une requête HTTP vers le serveur.

La réponse est fournie par celui-ci au navigateur.

À l'aide du HTML et des feuilles de style CSS, le résultat est ensuite affiché dans le navigateur.

La propriété XMLHttpRequest.readyState renvoie l'état dans lequel se trouve un client XMLHttpRequest. Un client XHR existe dans l'un des états suivants:

Les états possibles du "readyState" :

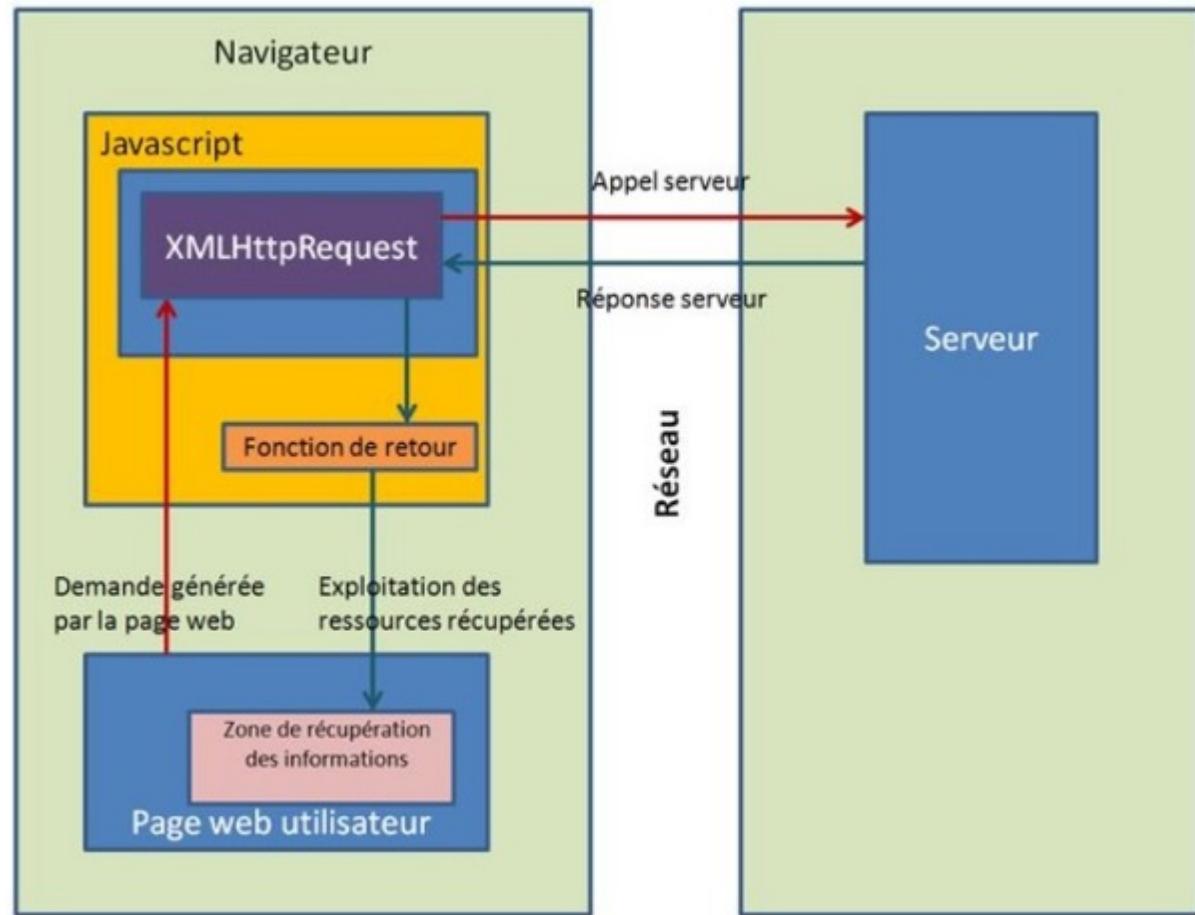
0 : Requête non initialisée, le client XMLHttpRequest a été créé, mais la méthode open() n'a pas encore été appelée.

1 : Connexion au serveur établie, la méthode open() a été invoquée. Pendant cet état, les en-têtes de demande peuvent être définis à l'aide de la méthode setRequestHeader() et la méthode send() peut être appelée, ce qui lancera la récupération.

2 : Requête reçue,

3 : Requête en cours de traitement

4 : Requête terminée et réponse prête.



Créer un objet XMLHttpRequest

Tous les navigateurs modernes (Chrome, Firefox, Edge (et IE7+), Safari, Opera) ont un objet XMLHttpRequest intégré.

Syntaxe pour créer un objet XMLHttpRequest :

```
variable = new XMLHttpRequest();
```

Méthodes d'objet XMLHttpRequest

Méthode	Description
new XMLHttpRequest()	Créer un nouvel objet XMLHttpRequest
abort()	Quitter la requête courante
getAllResponseHeaders()	Retourner toutes les informations du header
getResponseHeader()	Retourner une information spécifique du header
open(<i>method, url, async, user, psw</i>)	Specifier la requête <i>method</i> : GET ou POST <i>url</i> : emplacement du fichier <i>async</i> : true (asynchronous) ou false (synchronous) <i>user</i> : nom d'utilisateur (optionnel) <i>psw</i> : mot de passe (optionnel)
send()	Envoyer la requête au serveur (Utilisé dans les requêtes GET)
send(<i>string</i>)	Envoyer la requête au serveur (Utilisé dans les requêtes POST)

Propriétés de l'objet XMLHttpRequest

Propriété	Description
onreadystatechange	Définit une fonction à appeler lorsque la propriété readyState change
readyState	Contient le statut de XMLHttpRequest. 0 : requête non initialisée 1 : connexion serveur établie 2 : requête reçue 3 : traitement requête 4 : requête terminée et réponse prête
responseText	Renvoie les données de réponse sous forme de chaîne
responseXML	Renvoie les données de réponse sous forme de données XML
status	Renvoie le numéro d'état d'une requête 200: "OK" 403: "Forbidden" 404: "Not Found"
statusText	Renvoie le texte d'état (par exemple "OK" ou "Not Found")

Envoyer une demande à un serveur

Pour envoyer une requête à un serveur, nous utilisons les méthodes open() et send() de l'objet XMLHttpRequest : `open(method, url, async)`

Exemple1

```
xhttp.open("GET", "ajax_info.php", true);
xhttp.send();
```

Exemple2

```
xhttp.open("GET", "demo_get2.php?fname=Hassan&lname=FILALI", true);
xhttp.send();
```

Pour POST des données comme un formulaire HTML, ajoutez un en-tête HTTP avec setRequestHeader(). Spécifiez les données que vous souhaitez envoyer dans la méthode send() :

Exemple3

```
xhttp.open("POST", "demo_post2.php", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Hassan&lname=FILALI");
```

Les requêtes du serveur doivent être envoyées de manière asynchrone.

Le paramètre `async` de la méthode `open()` doit être défini sur `true` :

En envoyant de manière asynchrone, le JavaScript n'a pas à attendre la réponse du serveur, mais peut à la place :

- exécuter d'autres scripts en attendant la réponse du serveur
- traiter la réponse une fois que la réponse est prête

La propriété **onreadystatechange**

Avec l'objet XMLHttpRequest, vous pouvez définir une fonction à exécuter lorsque la requête reçoit une réponse.

La fonction est définie dans la propriété **onreadystatechange** de l'objet **XMLHttpRequest** :

```
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        document.getElementById("demo").innerHTML = this.responseText;
    }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

La propriété onreadystatechange

La propriété readyState contient le statut de XMLHttpRequest.

La propriété onreadystatechange définit une fonction à exécuter lorsque readyState change.

La propriété status et la propriété statusText contiennent le statut de l'objet XMLHttpRequest.

Property	Description
onreadystatechange	Définit une fonction à appeler lorsque la propriété readyState change
readyState	Contient le statut de XMLHttpRequest. 0 : requête non initialisée 1 : connexion serveur établie 2 : requête reçue 3 : traitement requête 4 : requête terminée et réponse prête
status	200: "OK" 403: "Forbidden" 404: "Page not found"
statusText	Renvoie le texte d'état (par exemple "OK" ou "Not Found")

02 – DÉCOUVRIR AJAX

Fonctionnement d'AJAX



La propriété onreadystatechange

La fonction onreadystatechange est appelée chaque fois que readyState change.

Lorsque readyState est 4 et status est 200, la réponse est prête :

```
function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML =
            this.responseText;
        }
    };
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
}
```

Utilisation d'une fonction Callback

Une fonction de rappel est une fonction passée en paramètre à une autre fonction.

Si vous avez plusieurs tâches AJAX sur un site Web, vous devez créer une fonction pour exécuter l'objet XMLHttpRequest et une fonction de rappel pour chaque tâche AJAX.

L'appel de fonction doit contenir l'URL et la fonction à appeler lorsque la réponse est prête.

```
<div id="demo">  
  
<h1>The XMLHttpRequest Object</h1>  
  
<button type="button"  
onclick="loadDoc('ajax_info.txt', myFunction)">  
Change Content  
</button>  
</div>  
  
<script>  
    ←  
</script>
```

```
function loadDoc(url, cFunction) {  
    var xhttp;  
    xhttp=new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            cFunction(this);  
        }  
    };  
    xhttp.open("GET", url, true);  
    xhttp.send();  
}  
  
function myFunction(xhttp) {  
    document.getElementById("demo").innerHTML =  
    xhttp.responseText;  
}
```



CHAPITRE 2

DÉCOUVRIR AJAX

1. Introduction à AJAX
2. Fonctionnement d'AJAX
3. Implémentation d'AJAX via jQuery

La méthode jQuery ajax()

La méthode jQuery ajax() fournit les fonctionnalités de base d'Ajax dans jQuery. Il envoie des requêtes HTTP asynchrones au serveur.

Syntaxe

```
$.ajax(url, [options]);
```

Description des paramètres :

url : URL de chaîne à laquelle vous souhaitez soumettre ou récupérer les données

options : options de configuration pour la requête Ajax. Un paramètre d'options peut être spécifié à l'aide du format JSON. Ce paramètre est facultatif.

Options	Description
accepte	Le type de contenu envoyé dans l'en-tête de requête qui indique au serveur quel type de réponse il acceptera en retour.
asynchrone	Par défaut, toutes les demandes sont envoyées de manière asynchrone. Définissez-le sur false pour le rendre synchrone.
avantEnvoyer	Une fonction de rappel à exécuter avant l'envoi de la requête Ajax.
Cache	Un booléen indiquant le cache du navigateur. La valeur par défaut est true.
complete	Une fonction de rappel à exécuter lorsque la demande se termine.

La méthode jQuery ajax()

Options	Description
contentType	Une chaîne contenant un type de contenu lors de l'envoi de contenu MIME au serveur. La valeur par défaut est "application/x-www-form-urlencoded; charset=UTF-8"
crossDomain	Une valeur booléenne indiquant si une requête est un interdomaine.
data	Une donnée à envoyer au serveur. Il peut s'agir d'un objet JSON, d'une chaîne ou d'un tableau.
dataType	Le type de données que vous attendez du serveur.
error	Une fonction de rappel à exécuter lorsque la requête échoue.
global	Un booléen indiquant s'il faut ou non déclencher un gestionnaire de requêtes Ajax global. La valeur par défaut est true.
headers	Un objet de paires clé/valeur d'en-tête supplémentaires à envoyer avec la demande.
ifModified	Autorisez la réussite de la demande uniquement si la réponse a changé depuis la dernière demande. Cela se fait en vérifiant l'en-tête Last-Modified. La valeur par défaut est false.
isLocal	Permettre à l'environnement actuel d'être reconnu comme local.
jsonp	Remplacez le nom de la fonction de rappel dans une requête JSONP. Cette valeur sera utilisée à la place de 'callback' dans le 'callback=?' partie de la chaîne de requête dans l'url.
jsonpCallback	Chaîne contenant le nom de la fonction de rappel pour une requête JSONP.

La méthode jQuery ajax()

Options	Description
mimeType	Chaîne contenant un type mime pour remplacer le type mime XMLHttpRequest.
password	Un mot de passe à utiliser avec XMLHttpRequest en réponse à une demande d'authentification d'accès HTTP.
processData	Un booléen indiquant si les données affectées à l'option de données seront converties en une chaîne de requête. La valeur par défaut est true.
statusCode	Un objet JSON contenant des codes HTTP numériques et des fonctions à appeler lorsque la réponse a le code correspondant.
success	Une fonction de rappel à exécuter lorsque la requête Ajax réussit.
timeout	Une valeur numérique en millisecondes pour le délai d'expiration de la demande.
type	Un type de requête http, par exemple POST, PUT et GET. La valeur par défaut est GET.
url	Une chaîne contenant l'URL à laquelle la demande est envoyée.
username	Un nom d'utilisateur à utiliser avec XMLHttpRequest en réponse à une demande d'authentification d'accès HTTP.
xhr	Un rappel pour créer l'objet XMLHttpRequest.
xhrFields	Un objet de paires fieldName-fieldValue à définir sur l'objet XMLHttpRequest natif.

Envoyer une demande Ajax

Les méthodes ajax() effectuent une requête http asynchrone et récupèrent les données du serveur. L'exemple suivant montre comment envoyer une simple requête Ajax

```
$ .ajax('/jquery/getdata', // request url
{
    success: function (data, status, xhr) { // success callback function
        $('p').append(data);
    }
});

<p></p>
```

Dans l'exemple ci-dessus, le premier paramètre '/getData' de la méthode ajax() est une URL à partir de laquelle nous voulons récupérer les données.

Par défaut, la méthode ajax() exécute la requête http GET si le paramètre d'option n'inclut pas l' option de méthode .

Le deuxième paramètre est le paramètre options au format JSON où nous avons spécifié la fonction de rappel qui sera exécutée lorsque la demande aboutira. Vous pouvez configurer d'autres options comme indiqué dans le tableau ci-dessus

Envoyer une demande Ajax

Les méthodes ajax() effectuent une requête http asynchrone et récupèrent les données du serveur. L'exemple suivant montre comment envoyer une simple requête Ajax

```
$ .ajax('/jquery/getdata', // request url
{
    success: function (data, status, xhr) { // success callback function
        $('p').append(data);
    }
});

<p></p>
```

Dans l'exemple ci-dessus, le premier paramètre '/getdata' de la méthode ajax() est une URL à partir de laquelle nous voulons récupérer les données.

Par défaut, la méthode ajax() exécute la requête http GET si le paramètre d'option n'inclut pas l' option de méthode .

Le deuxième paramètre est le paramètre options au format JSON où nous avons spécifié la fonction de rappel qui sera exécutée lorsque la demande aboutira. Vous pouvez configurer d'autres options comme indiqué dans le tableau ci-dessus

Envoyer une demande Ajax

L'exemple suivant montre comment obtenir les données JSON à l'aide de la méthode ajax().

```
$.ajax('/jquery/getjsondata',
{
    dataType: 'json', // type of response data
    timeout: 500,    // timeout milliseconds
    success: function (data,status,xhr) { // success callback function
        $('p').append(data.firstName + ' ' + data.middleName + ' ' + data.lastName);
    },
    error: function (jqXhr, textStatus, errorMessage) { // error callback
        $('p').append('Error: ' + errorMessage);
    }
});

<p></p>
```

Dans l'exemple ci-dessus, le premier paramètre est une URL de requête qui renverra des données JSON. Dans le paramètre options, nous avons spécifié les options dataType et timeout. L'option dataType spécifie le type de données de réponse, dans ce cas il s'agit de JSON. Le paramètre timeout spécifie le délai d'expiration de la demande en millisecondes. Nous avons également spécifié des fonctions de rappel pour l'erreur et le succès.

Envoyer une demande Ajax

La méthode ajax() renvoie un objet de jQuery XMLHttpRequest. L'exemple suivant montre comment utiliser l'objet jQuery XMLHttpRequest.

```
var ajaxReq = $.ajax('GetJsonData', {
    dataType: 'json',
    timeout: 500
});

ajaxReq.success(function (data, status, jqXhr) {
    $('p').append(data.firstName + ' ' + data.middleName + ' ' + data.lastName);
})

ajaxReq.error(function (jqXhr, textStatus, errorMessage) {
    $('p').append('Error: ' + errorMessage);
})

<p></p>
```

Dans l'exemple ci-dessus, le premier paramètre est une URL de requête qui renverra des données JSON. Dans le paramètre options, nous avons spécifié les options dataType et timeout. L'option dataType spécifie le type de données de réponse, dans ce cas il s'agit de JSON. Le paramètre timeout spécifie le délai d'expiration de la demande en millisecondes. Nous avons également spécifié des fonctions de rappel pour l'erreur et le succès.

Envoyer une requête HTTP POST en utilisant ajax()

La méthode ajax() peut envoyer tout type de requêtes http. L'exemple suivant envoie une requête HTTP POST au serveur.

```
$.ajax('/jquery/submitData', {
    type: 'POST', // http method
    data: { myData: 'This is my data.' }, // data to submit
    success: function (data, status, xhr) {
        $('p').append('status: ' + status + ', data: ' + data);
    },
    error: function (jqXhr, textStatus, errorMessage) {
        $('p').append('Error' + errorMessage);
    }
});
```

Dans l'exemple ci-dessus, le premier paramètre est une URL qui est utilisée pour soumettre les données. Dans le paramètre options, nous avons spécifié une option de type en tant que POST, donc la méthode ajax() enverra une requête HTTP POST. De plus, nous avons spécifié l'option de données en tant qu'objet JSON contenant des données qui seront soumises au serveur.

Ainsi, vous pouvez envoyer une requête GET, POST ou PUT en utilisant la méthode ajax().

Méthode jQuery get()

La méthode jQuery get() envoie une requête http GET asynchrone au serveur et récupère les données.

Syntaxe `$.get(url, [données],[rappel]);`

Description des paramètres :

url : url de la requête à partir de laquelle vous souhaitez récupérer les données

data : données à envoyer au serveur avec la requête comme chaîne de requête

callback : fonction à exécuter lorsque la requête aboutit

L'exemple suivant montre comment récupérer des données à partir d'un fichier texte.

```
$.get('/data.txt', // url
      function (data, textStatus, jqXHR) { // success callback
          alert('status: ' + textStatus + ', data:' + data);
      });

```

Dans l'exemple ci-dessus, le premier paramètre est une URL à partir de laquelle nous voulons récupérer les données. Ici, nous voulons récupérer les données d'un fichier txt situé sur mydomain.com/data.txt. Veuillez noter que vous n'avez pas besoin de donner l'adresse de base.

Méthode jQuery get()

L'exemple suivant montre comment récupérer des données JSON à l'aide de la méthode get().

```
$ .get( '/jquery/getjsondata', {name: 'Hassan'}, function  
  (data, textStatus, jqXHR) {  
    $('p').append(data.firstName);  
  };  
  
<p></p>
```

Dans l'exemple ci-dessus, le premier paramètre est une URL à partir de laquelle nous voulons récupérer les données JSON. Cette URL peut être un service Web ou toute autre URL qui renvoie des données au format JSON.

Le deuxième paramètre correspond aux données à envoyer au serveur sous forme de chaîne de requête. Nous avons spécifié le paramètre de nom avec la valeur 'Steve' au format JSON. Alors maintenant, l'url de la requête ressemblerait à <http://mydomain.com/jquery/getjsondata?name=Hassan>

Le troisième paramètre est une fonction de rappel qui sera exécutée lorsque cette requête GET réussit.

Méthode jQuery getJSON()

La méthode jQuery getJSON() envoie une requête http GET asynchrone au serveur et récupère les données au format JSON en définissant accepte l'en-tête sur application/json, text/javascript. C'est la même chose que la méthode get(), la seule différence est que la méthode getJSON() récupère spécifiquement les données JSON tandis que la méthode get() récupère tout type de données. C'est comme une méthode de raccourci pour récupérer les données JSON.

Syntaxe:

Description du paramètre :

url : url de la requête à partir de laquelle vous souhaitez récupérer les données

data : données JSON à envoyer au serveur sous forme de chaîne de requête

callback : fonction à exécuter lorsque la requête aboutit

L'exemple suivant montre comment récupérer des données JSON à l'aide de la méthode getJSON().

```
$.getJSON('/jquery/getjsondata', {name: 'Steve'}, function
(data, textStatus, jqXHR){
    $('p').append(data.firstName);
});

<p></p>
```

Méthode jQuery getJSON()

Vous pouvez attacher les méthodes de rappel fail et done à la méthode getJson() comme indiqué ci-dessous.

```
$getJSON('/jquery/getjsondata', { name:'Steve'}, function(data, textStatus, jqXHR){  
    alert(data.firstName);  
})  
.done(function () { alert('Request done!'); })  
.fail(function (jqxhr,settings,ex) { alert('failed, '+ ex);});
```

Méthode jQuery post()

La méthode jQuery post() envoie une requête HTTP POST asynchrone au serveur pour soumettre les données au serveur et obtenir la réponse.

Syntaxe: `$.post(url,[données],[rappel],[type]);`

Description du paramètre :

url : URL de demande à partir de laquelle vous souhaitez soumettre et récupérer les données.

data : données json à envoyer au serveur avec la requête sous forme de données de formulaire.

callback : fonction à exécuter lorsque la requête aboutit.

type : type de données du contenu de la réponse.

Voyons comment soumettre des données et obtenir la réponse à l'aide de la méthode post(). Considérez l'exemple suivant.

```
$.post('/jquery/submitData', // url
       { myData: 'This is my data.' }, // data to be submit
       function(data, status, jqXHR) { // success callback
           $('p').append('status: ' + status + ', data: ' + data);
       }
)
<p></p>
```

Méthode jQuery post()

L'exemple suivant montre comment soumettre et récupérer des données JSON à l'aide de la méthode post().

```
$ .post( '/submitJSONData', // url
          { myData: 'This is my data.' }, // data to be submit
          function(data, status, xhr) { // success callback function
              alert('status: ' + status + ', data: ' + data.responseText);
          },
          'json'); // response data format
```

Dans l'exemple ci-dessus, veuillez noter que le dernier paramètre est un type de données de réponse. Nous obtiendrons les données JSON en tant que réponse du serveur. Ainsi, la méthode post() analysera automatiquement la réponse dans l'objet JSON. Le reste des paramètres est identique au premier exemple.

Méthode jQuery post()

L'exemple suivant montre comment soumettre et récupérer des données JSON à l'aide de la méthode post().

```
$.post('/submitJSONData', // url
       { myData: 'This is my data.' }, // data to be submit
       function(data, status, xhr) { // success callback function
           alert('status: ' + status + ', data: ' + data.responseText);
       },
       'json'); // response data format
```

Dans l'exemple ci-dessus, veuillez noter que le dernier paramètre est un type de données de réponse. Nous obtiendrons les données JSON en tant que réponse du serveur. Ainsi, la méthode post() analysera automatiquement la réponse dans l'objet JSON. Le reste des paramètres est identique au premier exemple.

Vous pouvez également attacher les méthodes de rappel fail et done à la méthode post() comme indiqué ci-dessous.

```
$.post('/jquery/submitData',
       { myData: 'This is my data.' },
       function(data, status, xhr) {
           $('p').append('status: ' + status + ', data: ' + data);

       }).done(function() { alert('Request done!'); })
       .fail(function(jqxhr, settings, ex) { alert('failed, ' + ex); });

<p></p>
```

Méthode jQuery load()

La méthode jQuery load() permet de charger du contenu HTML ou texte à partir d'un serveur et de l'ajouter dans un élément DOM.

Syntaxe `$.load(url,[données],[rappel]);`

Description des paramètres :

url : url de la requête à partir de laquelle vous souhaitez récupérer le contenu

data : données JSON à envoyer avec requête au serveur.

callback : fonction à exécuter lorsque la requête aboutit

L'exemple suivant montre comment charger du contenu html depuis le serveur et l'ajouter à l'élément div.

```
$( '#msgDiv' ).load( '/demo.html' );  
  
<div id="msgDiv"></div>
```

Dans l'exemple ci-dessus, nous avons spécifié le fichier html à charger depuis le serveur et à ajouter son contenu à l'élément div.

Note : Si aucun élément n'est trouvé par le sélecteur alors la requête Ajax ne sera pas envoyée.

Méthode jQuery load()

La méthode load() nous permet de spécifier une partie du document de réponse à insérer dans l'élément DOM. Cela peut être réalisé à l'aide du paramètre url, en spécifiant un sélecteur avec une URL séparée par un ou plusieurs caractères d'espacement, comme illustré dans l'exemple suivant.

```
$('#msgDiv').load('/demo.html #myHtmlContent');

<div id="msgDiv"></div>
```

Dans l'exemple ci-dessus, le contenu de l'élément dont l'ID est myHtmlContent sera ajouté à l'élément msgDiv. Ce qui suit est un demo.html.

```
<!DOCTYPE html>
<html
  xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title></title>
  </head>
  <body>
    <h1>This is demo html page.</h1>
    <div id="myHtmlContent">This is my
      html content.</div>
  </body>
</html>
```

La méthode load() nous permet également de spécifier les données à envoyer au serveur et de récupérer les données.

```
$('#msgDiv').load('getData', // url
  { name: 'bill' }, // data
  function(data, status, jqXHR) { //
    // callback function
    alert('data loaded')
  });

<div id="msgDiv"></div>
```