



WEBFORCE
BE THE CHANGE



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



WEBFORCE
BE THE CHANGE



1

LE GUIDE DE SOUTIEN

Il contient le résumé théorique et le manuel des travaux pratiques



2

LA VERSION PDF

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



3

DES CONTENUS TÉLÉCHARGEABLES

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



4

DU CONTENU INTERACTIF

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



5

DES RESSOURCES EN LIGNES

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



PARTIE 1

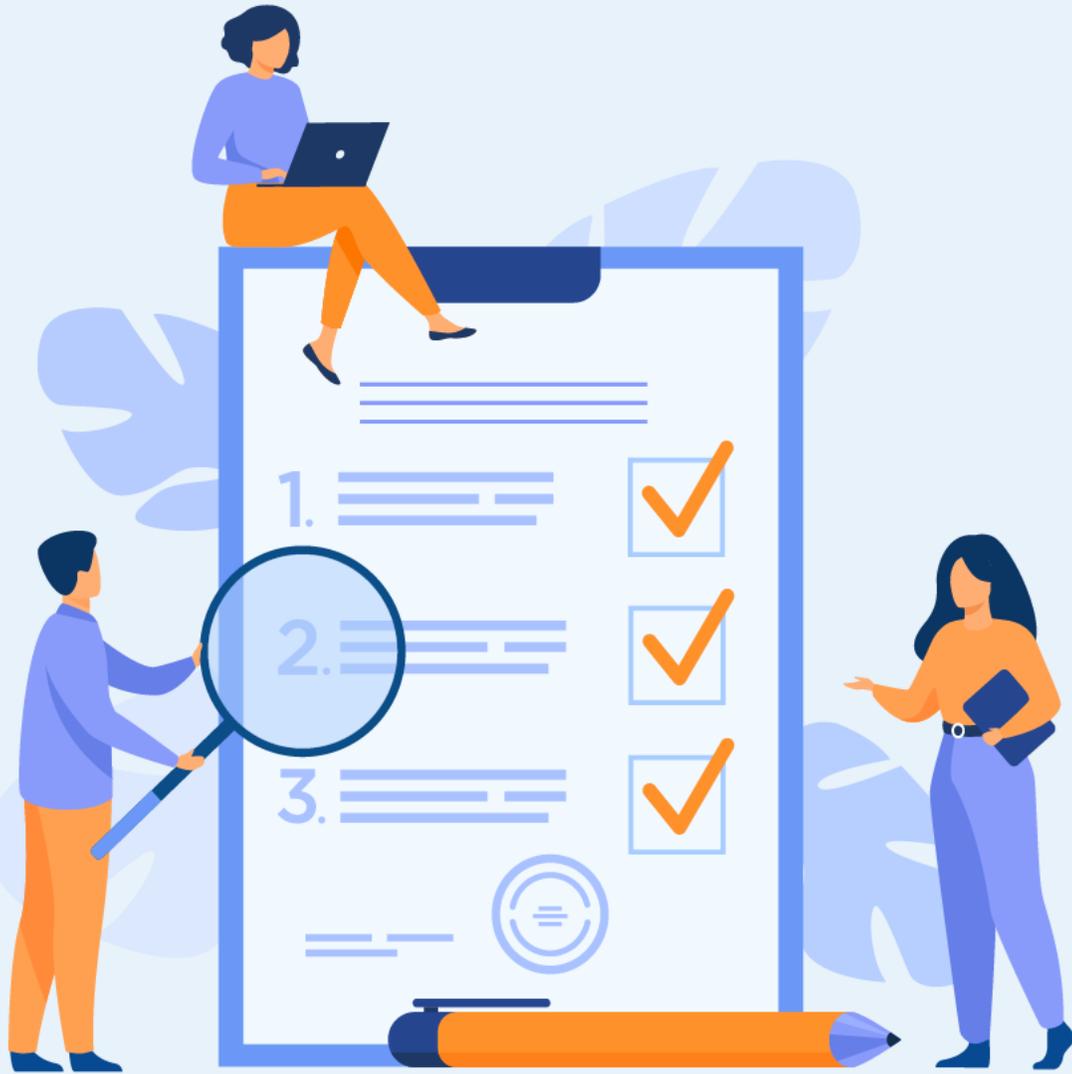
Définir le rôle de javascript dans le développement

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 compilé

Ce que vous allez apprendre dans ce chapitre :

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



04 heures

CHAPITRE 1

Comparer un langage de script avec un langage compilé

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



Notion de compilateur / interpréteur

- La compilation consiste à transformer le code écrit dans un langage de programmation de haut niveau (code lisible par l'homme) en code machine compréhensible par un processeur informatique (bits binaires 1 et 0). Le compilateur s'assure également que le programme est correct du point de vue TYPE : on n'est pas autorisé à affecter une chaîne de caractères à une variable entière. Le compilateur veille également à ce que votre programme soit syntaxiquement correct. Par exemple, "x * y" est valide, mais "x @ y" ne l'est pas.
- Un interpréteur est un programme informatique qui convertit chaque déclaration de programme de haut niveau en code machine. Cela inclut le code source, le code précompilé et les scripts.

Différence : Un compilateur convertit le code en code machine (crée un exe) avant l'exécution du programme. L'interpréteur convertit le code en code machine, ligne par ligne, au moment d'exécution du programme.

Exemples de langages compilés : C, C++, C#, CLEO, COBOL, etc.

Exemples de langages interprétés : JavaScript, Perl, Python, BASIC, etc.

01 - Langage de script vs langage compilé

Définir un langage de script

Langage compilé

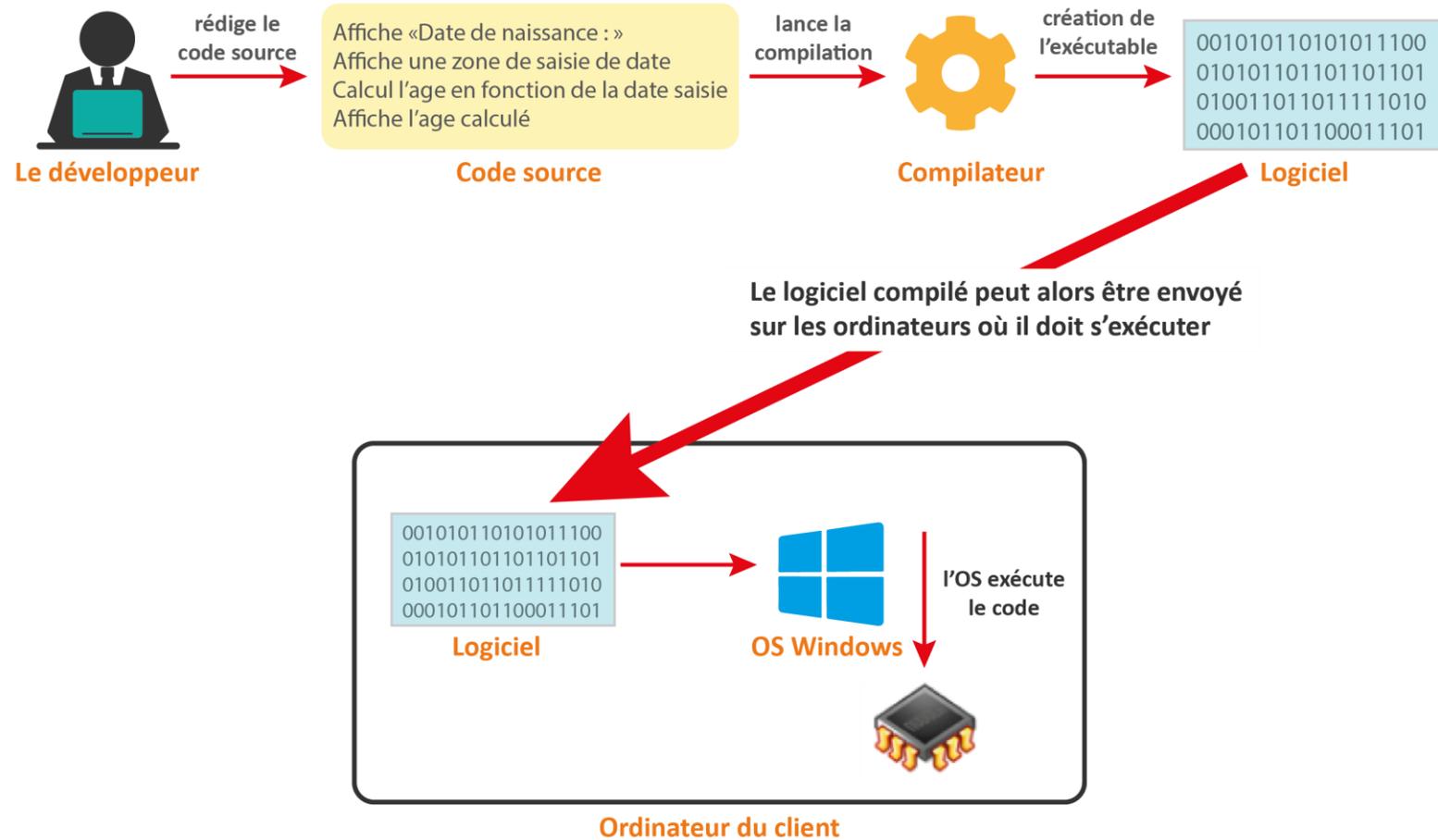


Figure 1 : Fonctionnement d'un langage compilé

01 - Langage de script vs langage compilé

Définir un langage de script

Langage interprété

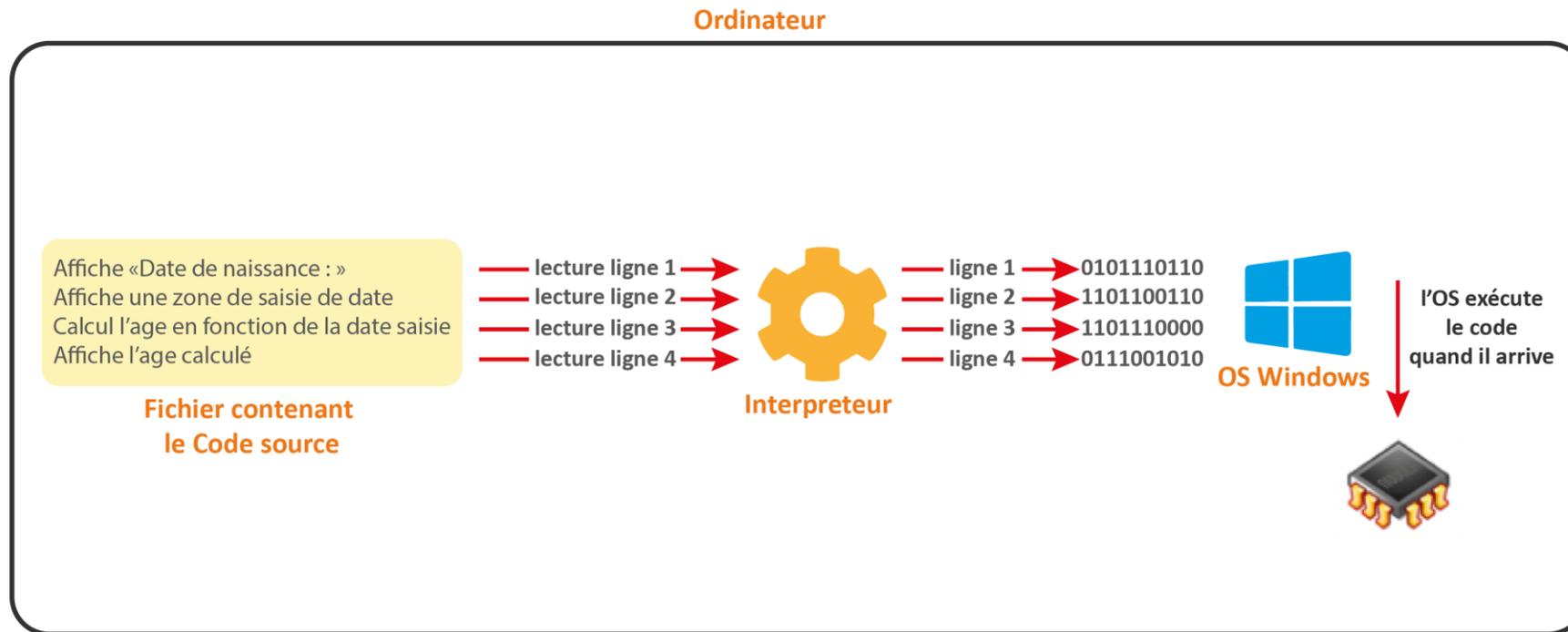


Figure 2 : Fonctionnement d'un langage interprété

01 - Langage de script vs langage compilé

Définir un langage de script



Langage de script

Un langage de script (également appelé script) est une série de commandes qui peuvent être exécutées sans compilation.

Tous les langages de script sont des langages de programmation, mais tous les langages de programmation ne sont pas des langages de script.

Les langages de script utilisent un programme appelé interpréteur pour traduire les commandes.

Types d'un langage de script : coté client et coté serveur

- **Les langages de script côté serveur** s'exécutent sur un serveur Web. Lorsqu'un client envoie une requête, le serveur répond en envoyant du contenu via le protocole HTTP. Les scripts côté serveur ne sont pas visibles par le public. Leur rôle est d'assurer la rapidité du traitement, l'accès aux données et la résolution des erreurs.
- Exemples de langages de script côté serveur :
 - **PHP, ASP .NET, Node.js, Java, Ruby, Perl.**
- **Les langages de script côté client** s'exécutent du côté du client, sur son navigateur Web. L'avantage des scripts côté client est qu'ils peuvent réduire la demande sur le serveur, ce qui permet aux pages Web de se charger plus rapidement. Ces scripts sont axés sur l'interface utilisateur et ses fonctionnalités.
- Exemples de langages de script côté client :
 - **HTML, CSS, JavaScript.**

CHAPITRE 1

Comparer un langage de script avec un langage compilé

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

Le rôle de l'interpréteur

- Le fonctionnement des langages de script est assuré par l'interpréteur. Son rôle réside dans la traduction des instructions du programme source en code machine.
- S'il y a une erreur dans l'instruction courante, l'interpréteur termine son processus de traduction à cette instruction et affiche un message d'erreur. L'interprète ne passe à la ligne d'exécution suivante qu'après avoir éliminé l'erreur.
- Un interpréteur exécute directement les instructions écrites dans un langage de script sans les convertir préalablement en code objet ou en code machine.

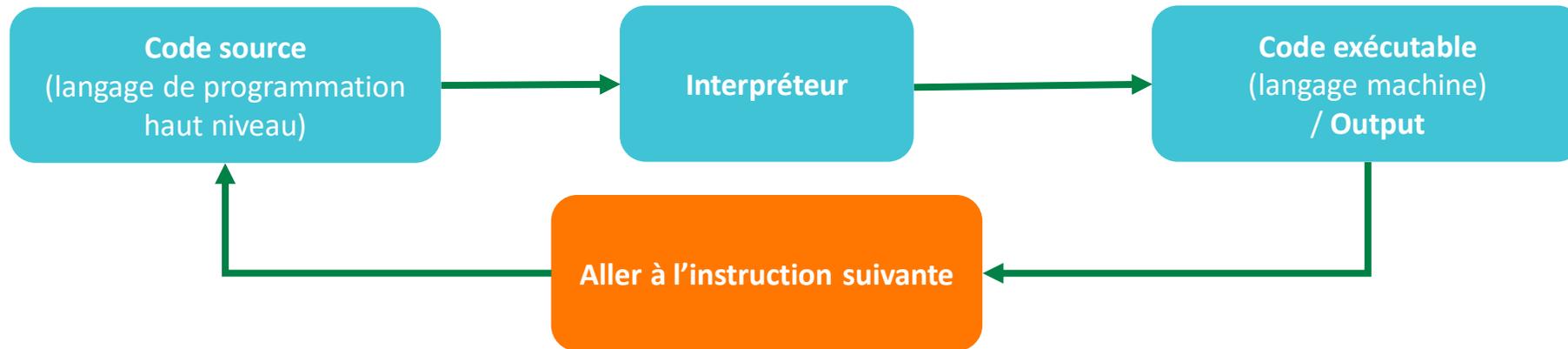
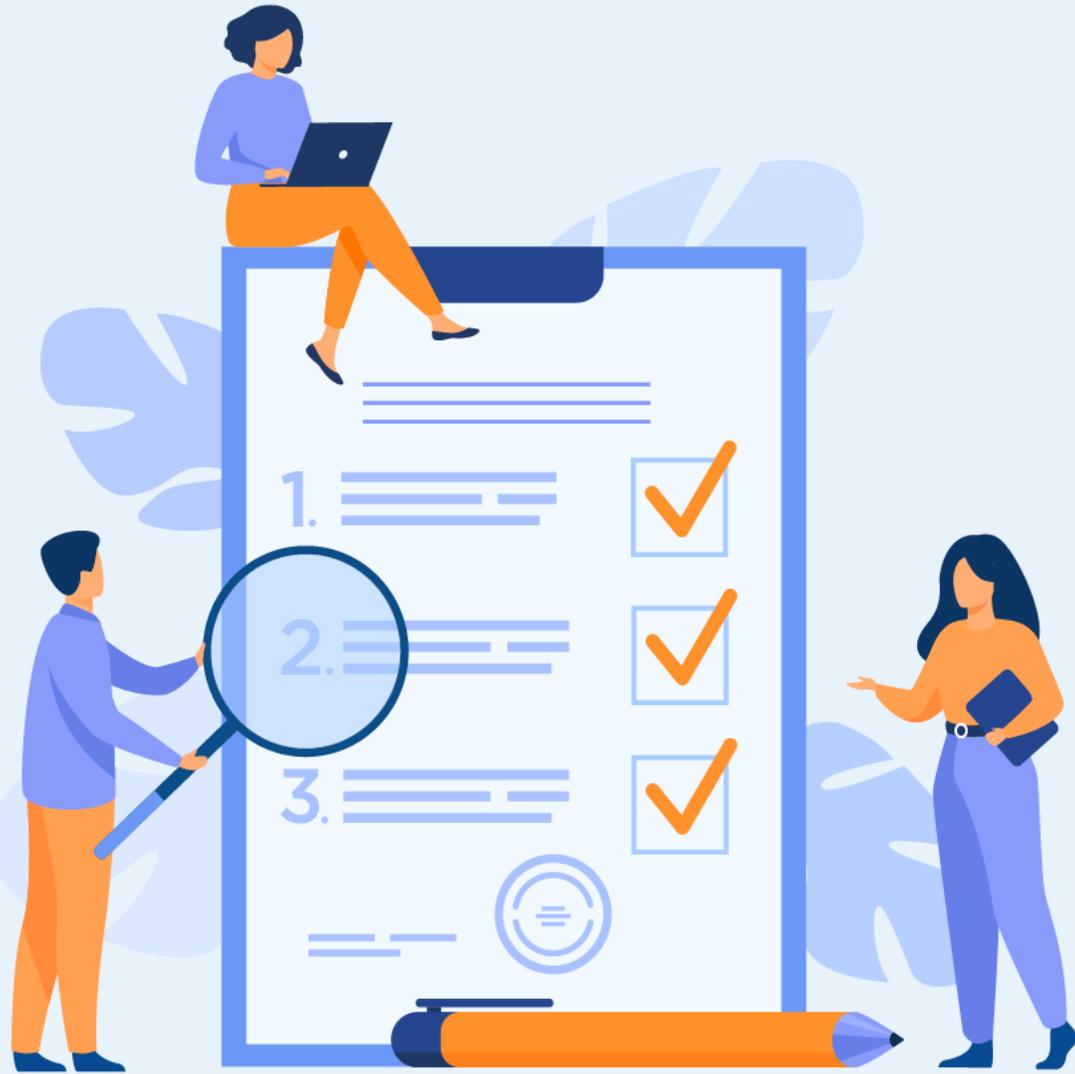


Figure 3: Fonctionnement d'un langage de script [1]



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



04 heures

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

- L'architecture client-serveur correspond à l'architecture d'un réseau informatique dans lequel de nombreux clients (processeurs distants) demandent et reçoivent des services d'un serveur centralisé (Serveur).
- Les **clients** sont souvent situés sur des postes de travail ou des ordinateurs personnels, tandis que les **serveurs** sont situés ailleurs sur le réseau, généralement sur des machines plus puissantes.

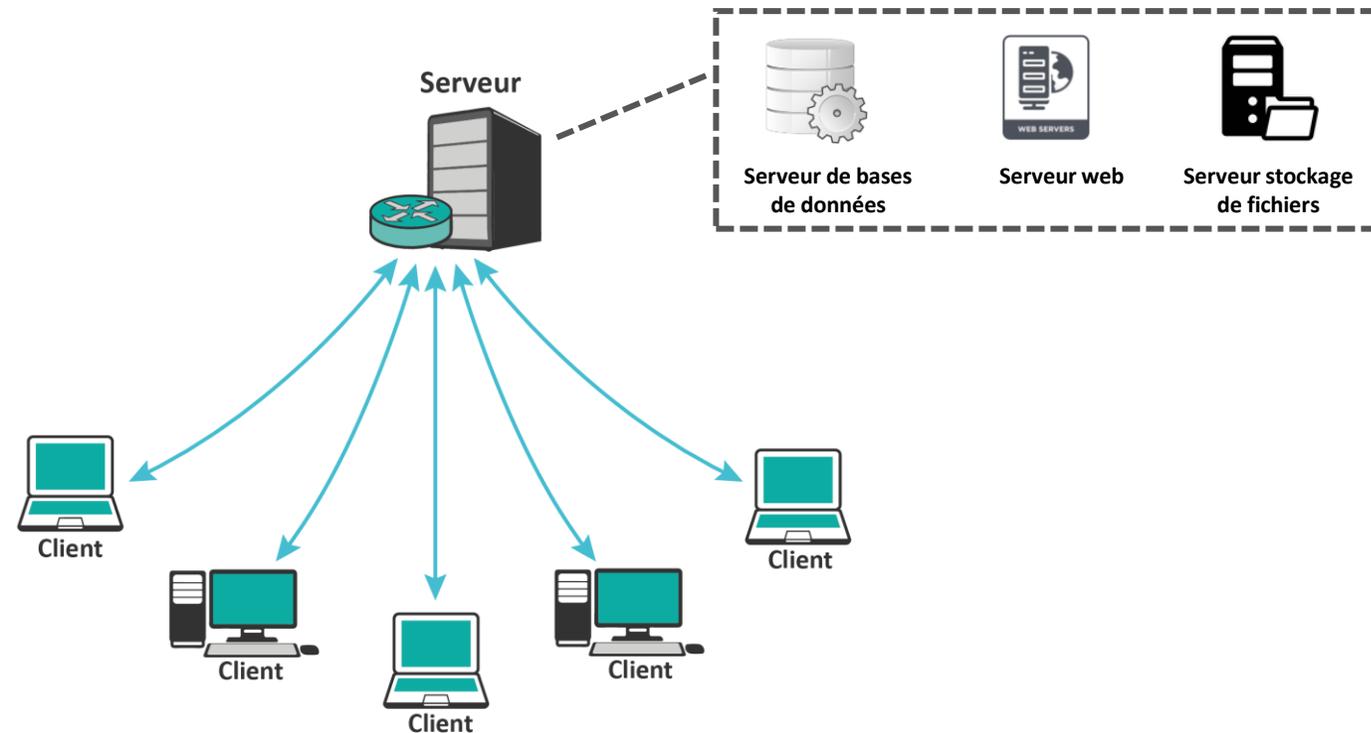


Figure 4 : Architecture Client-serveur [2]

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



Interaction Client / Serveur

Les ordinateurs clients fournissent une interface (comme les navigateur) permettant à un utilisateur de demander des services auprès de serveur et d'afficher les résultats. Cette interaction passe par les étapes suivantes :

- 1 L'utilisateur saisit l'URL (Uniform Resource Locator) du site web ou du fichier. Le navigateur demande alors au serveur le DNS (DOMAIN NAME SYSTEM).
- 2 Le serveur DNS recherche l'adresse du serveur WEB.
- 3 Le serveur DNS répond avec l'adresse IP du serveur Web.
- 4 Le navigateur envoie une requête HTTP/HTTPS à l'adresse IP du serveur WEB (fournie par le serveur DNS).
- 5 Le serveur envoie les fichiers nécessaires du site web (pages html, documents, images,).
- 6 Le navigateur rend alors les fichiers et le site web s'affiche. Ce rendu est effectué à l'aide de l'interpréteur DOM (Document Object Model), de l'interpréteur CSS et du moteur JS, collectivement connus sous le nom de compilateurs JIT ou (Just in Time).

02 - Comprendre l'architecture client/serveur

Fonctionnement d'un système client/serveur Web

Fonctionnement

Le fonctionnement d'un système client/serveur peut être illustré par le schéma suivant :

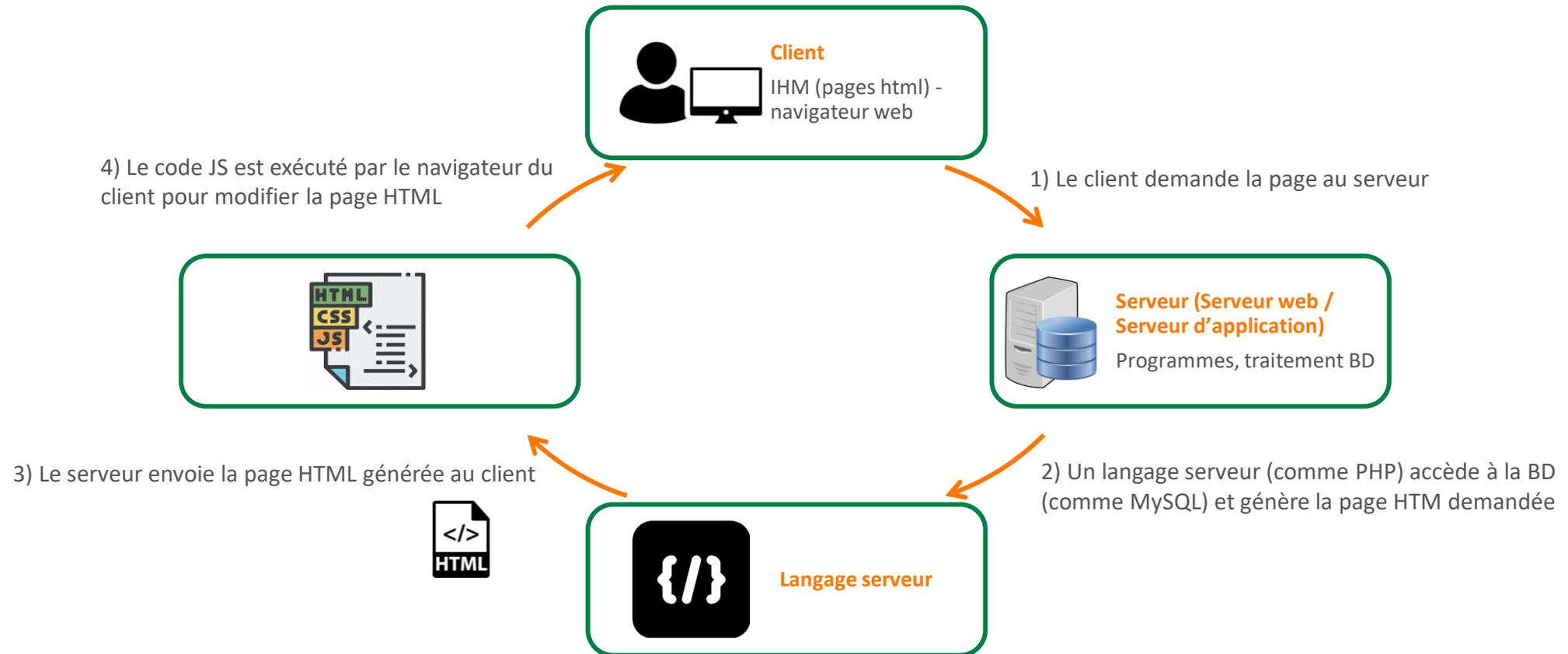


Figure 5 : Fonctionnement d'un système client serveur

02 - Comprendre l'architecture client/serveur

Fonctionnement d'un système client/serveur Web



Serveurs Web et HTTP

Les navigateurs web (clients) communiquent avec les serveurs web via le protocole HTTP (Hypertext Transfer Protocol). En tant que protocole de requête-réponse, ce protocole permet aux utilisateurs d'interagir avec des ressources Web telles que des fichiers HTML en transmettant des messages hypertextes entre les clients et les serveurs. Les clients HTTP utilisent généralement des connexions TCP (Transmission Control Protocol) pour communiquer avec les serveurs.

Une requête HTTP inclut :

- Une URL pointant sur la cible et la ressource (un fichier HTML, un document, ...).
- Une méthode de requête spécifique afin d'effectuer diverses tâches (par exemple mise à jour des données, récupération d'un fichier, ...).

Les différentes méthodes de requêtes et les actions associées sont présentées dans le tableau ci-dessous :

Méthode	Rôle
GET	Récupération d'une ressource spécifique (fichier html par exemple).
POST	Création d'une nouvelle ressource, enregistrement des données d'un formulaire d'inscription...
HEAD	Récupération des informations "metadata" d'une ressource spécifique sans le "body" .
PUT	Met à jour une ressource existante ou en crée une si elle n'existe pas.
DELETE	Suppression la ressource spécifiée.

02 - Comprendre l'architecture client/serveur

Fonctionnement d'un système client/serveur Web



Serveurs Web et HTTP

- La réponse HTTP (**HTTP Response**) est l'information fournie par le serveur suite à la demande du client. Elle sert à confirmer que l'action demandée a été exécutée avec succès. En cas d'erreur dans l'exécution de la demande du client, le serveur répond par un message d'erreur.
- Les réponses HTTP se présentent sous la forme d'un texte brut formaté au format JSON ou XML, tout comme les demandes HTTP. Le corps d'une réponse aboutie à une requête GET contiendrait la ressource demandée.

Exemples de code d'état HTTP (HTTP status codes) :

- **"200 OK"** : succès
- **"404 Not Found"** : ressource introuvable
- **"403 Forbidden"** : accès non autorisé

02 - Comprendre l'architecture client/serveur

Fonctionnement d'un système client/serveur Web



Serveurs Web et HTTP

Exemple de réponse HTTP (clic sur le logo du site www.ofppt.ma en utilisant le navigateur Google Chrome) :

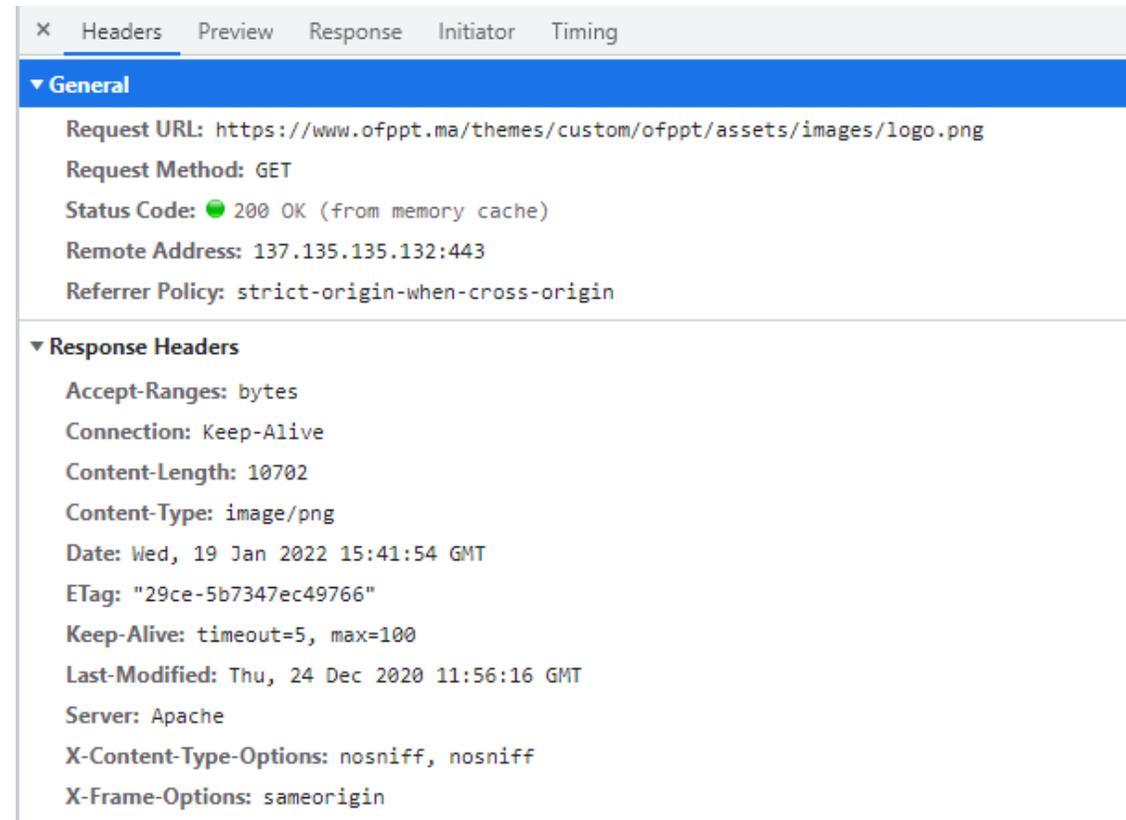


Figure 6: Exemple de réponse HTTP

CHAPITRE 3

Découvrir l'écosystème de développement

Ce que vous allez apprendre dans ce chapitre :

- Environnements de développement
- Découverte des bibliothèques appropriées (jQuery, React, Vue JS, Angular,...)



04 heures

CHAPITRE 3

Découvrir l'écosystème de développement

1. **Environnements de développement**
2. Découverte des bibliothèques appropriées (jQuery, React, Vue JS, Angular, ...)



Choix de l'environnement de développement

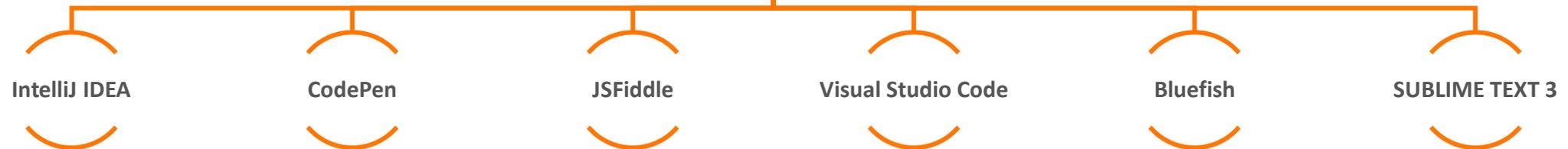
Un Environnement de Développement Intégré (Integrated development environment– IDE an anglais) désigne un outil de développement dit tout en un qui permet aux programmeurs de consolider les différents aspects de l'écriture d'un programme informatique.

Les IDE assistent les programmeurs dans l'édition des logiciels en combinant les activités courantes de programmation en une seule application :

- Édition du code source
- Mise en évidence de la syntaxe (colorisation)
- Saisie automatique (Auto-complétion)
- Création d'exécutables
- Débogage

IDE pour le web :

(Liste complète et lien de téléchargement disponible sur <https://www.guru99.com/web-development-ide.html>)



03 - Découvrir l'écosystème de développement

Environnements de développement



Rappel : Visual Studio Code

Comme discuté dans M104, nous utiliserons le logiciel **Visual Studio Code** qui est un logiciel gratuit qui permet l'édition, la correction et le débogage du code source dans plusieurs langages informatiques : Visual Basic, JavaScript, XML, Python, HTML, CSS,

VS code offre :

- Une présentation sophistiquée du code.
- Une auto-complétion du code source.
- Un ensemble d'extensions permettant de simplifier la programmation.
- La détection du langage de programmation par l'extension du fichier.

CHAPITRE 3

Découvrir l'écosystème de développement

1. Environnements de développement
2. **Découverte des bibliothèques appropriées (jQuery, React, Vue JS, Angular, ...)**



03 - Découvrir l'écosystème de développement

Découverte des bibliothèques appropriées



Front-end vs back-end

Front-End

- Le terme "**front-end**" désigne l'interface utilisateur.
- Le front-end est construit en utilisant une combinaison de technologies telles que le langage de balisage hypertexte (HTML), JavaScript et les feuilles de style en cascade (CSS).

Back-End

- Le terme "**back-end**" désigne le serveur, l'application et la base de données qui travaillent en coulisses pour fournir des informations à l'utilisateur.
- La programmation back-end est définie comme la logique informatique d'un site Web ou d'un logiciel, depuis le stockage et l'organisation des données jusqu'à la création des algorithmes et des séquences logiques complexes qui fonctionnent, d'une manière transparente, sur le front-end.
- Les langages back-end les plus populaires sont Ruby, Python, Java, ASP .Net et PHP.

Les frameworks front-end

1. Angular JS
2. React.js
3. JQuery
4. Vue.js



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



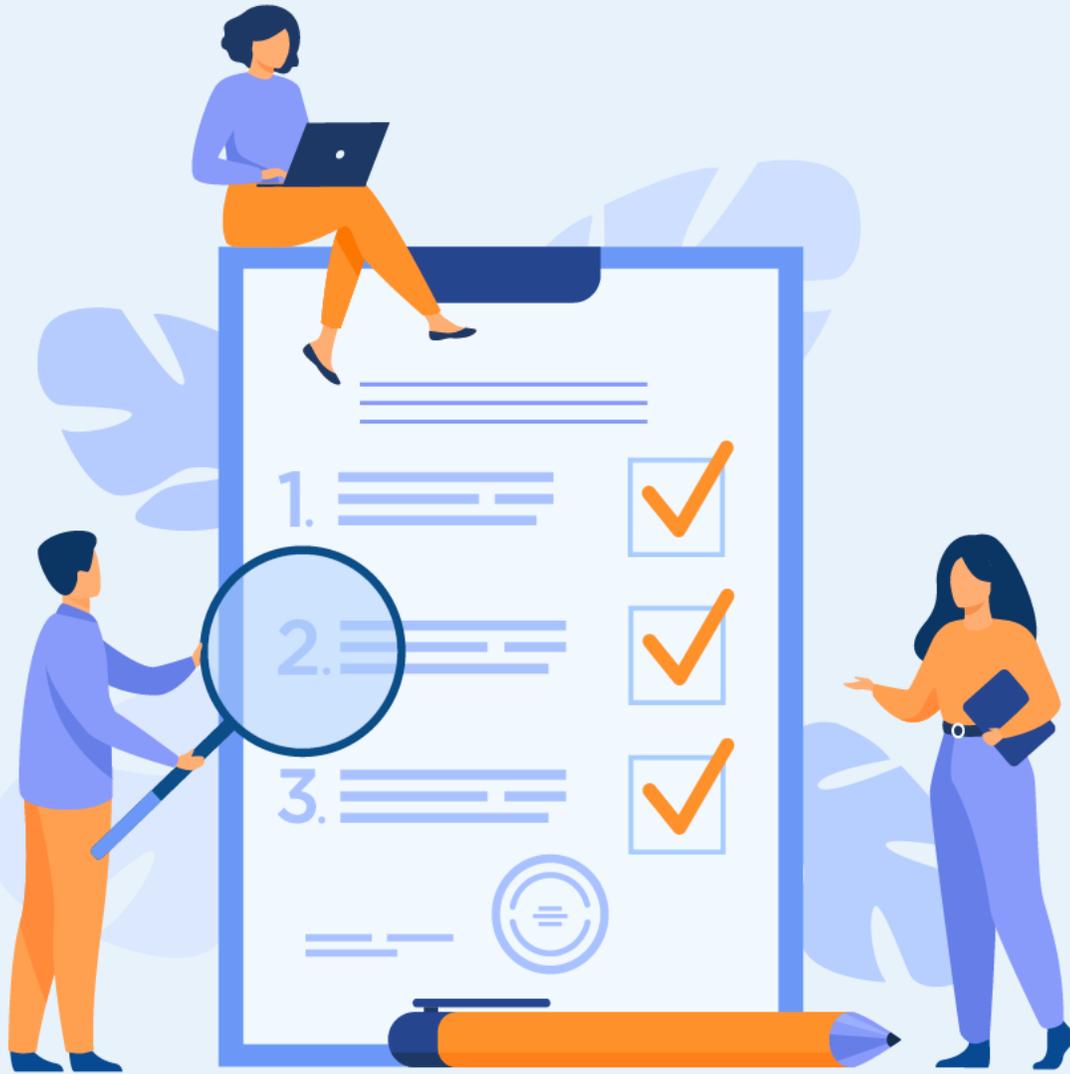
48 heures

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

Introduction



Introduction à JavaScript

- Javascript est un langage de scripts qui peut être incorporé aux balises Html. JS est exécuté par le navigateur ;
- Son rôle est d'améliorer la présentation et l'interactivité des pages Web ;
- JavaScript offre des « gestionnaires d'événement » qui reconnaissent et réagissent aux demandes du client (comme les mouvements de la souris, clics sur les touches du clavier, etc.)



Remarques

- JS est un langage de Script dépendant de HTML.
- Code interprété par le browser au moment de l'exécution à la volée (Just In Time).
- JS Permet d'accéder aux objets du navigateur.
- JavaScript est « case sensitive ».

Intégration de JavaScript dans HTML

JavaScript peut être intégré n'importe où dans le document HTML. Il est aussi possible de l'utiliser plusieurs fois dans le même document HTML.

- **Méthode 1 : Code JavaScript intégré au document html**

```
<script language="JavaScript">  
  
/* ou // code js*/  
  
</script>
```

- **Méthode 2 : Code JavaScript externe**

```
<script language="javascript" src="monScript.js"> </script>
```

- **Méthode 3 : Pseudo-URL**

```
<a href="JavaScript:window.alert('Welcome to JavaScript!');">clickez ici</a>
```

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Notions de variables et de données



Identifiants JavaScript

- Un identifiant en JS correspond au nom d'une variable. Ce nom doit être unique.
- Ces identifiants peuvent être des noms courts (comme x et y) ou bien des noms plus descriptifs (comme note, total, NomComplet, etc.).

Les règles à respecter lors du choix des noms de variables sont :

- Un identifiant peut contenir des lettres, des chiffres, des traits de soulignement (_) et des signes dollar (\$).
- Un identifiant peut commencer par une lettre, un signe \$ ou bien un signe _
- JS est sensible à la casse (y et Y sont considérés comme des variables différentes).
- Un identifiant ne peut pas être un mot réservé du langage.

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Notions de variables et de données



Types de données JavaScript

JavaScript est un langage faiblement typé : le type d'une variable est défini au moment de l'exécution.

On peut déclarer une variable JS en utilisant les mots-clés suivants :

- **var** : utilisé pour déclarer une variable globale (function scope) ;
 - **let** : utilisé pour déclarer une variable dont la portée est limitée à un bloc (block scope) ;
 - **const** : permet de déclarer une variable qui doit avoir une valeur initiale et ne peut pas être réassignée.
- **Déclaration explicite (en utilisant le mot clé var) :**

```
var nom_variable = new Type de la variable;  
var nom_variable;
```
 - **Déclaration implicite (sans utiliser var, on écrit le nom de la variable suivie du caractère d'affectation et de la valeur à affecter)**

```
Numéro = 1 ; Prénom = "xyz" ;
```



Remarques

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

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Notions de variables et de données



Types de données JavaScript (exemples)

Déclaration des variable booléennes :

- `var test=new Boolean(true) ;`
- `test=new Boolean(1) ;`
- `let test = true.`

Déclaration des chaînes de caractères :

- `var chaine = "Bonjour";`

Déclaration des nombres :

- `var entier = 60; //un entier ;`
- `let pi = 3.14; //un nombre réel.`

Déclaration de plusieurs variables :

- `var variable3 = 2, variable4 = "mon texte d'initialisation";`

Déclaration sans initialisation :

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

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Notions de variables et de données



Types de données JavaScript (exemples)

Const permet de créer des variables JavaScript qui ne peuvent être ni redéclarées ni réaffectées (constantes). Ces variables doivent être initialisées à la déclaration.

Exemple :

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

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

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Notions de variables et de données



Portée des variables (variable scope)

La portée d'une variable détermine son accessibilité (visibilité). En JS, trois types de portées sont distinguées :

Portée du bloc : (Block scope)

- En utilisant le mot clé **let**, 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
```

- En utilisant le mot clé **var**, les variables déclarées à l'intérieur d'un bloc { } sont accessibles depuis l'extérieur du bloc.

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

Portée locale : (Function scope)

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

```
function Test()  
{  
    var x = "test1";  
    let y = "test2";  
    const z = "test3";  
}  
//x, y et z ne sont pas  
accessibles en dehors de la  
fonction
```

Portée globale : (Global scope)

- Une variable déclarée en dehors d'une fonction, devient GLOBAL. Les variables globales sont accessibles de n'importe où dans un programme JavaScript..

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



Opérateurs arithmétiques JavaScript

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

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
**	Puissance
/	Division
%	Modulo (Reste de la division)
++	Incrémement
--	Décrémement

Opérateurs d'affectation JavaScript

Les opérateurs d'affectation permettent d'assigner des valeurs aux variables JavaScript. Le tableau suivant regroupe ces opérateurs :

Opérateur	Exemple d'utilisation	Signification	Exemple complet	Résultat
=	x = y	x prend la valeur de y	let x = 5	x vaut 5
+=	x += y	x = x + y	let x = 10; x += 5;	x vaut 15
-=	x -= y	x = x - y	let x = 10; x -= 5;	x vaut 5
*=	x *= y	x = x * y	let x = 10; x *= 5;	x vaut 50
/=	x /= y	x = x / y	let x = 10; x /= 5;	x vaut 2
%=	x %= y	x = x % y	let x = 10; x %= 5;	x vaut 0
**=	x **= y	x = x ** y ⇔ x = x à la puissance y	let x = 3; x **= 2;	x vaut 9

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Expressions et opérateurs



Concaténation des chaînes de caractères en JavaScript

L'opérateur + , appliqué aux chaînes de caractères, permet de concaténer des chaînes.

Exemple 1 :

```
let texte1 = "OFPPT";
texte1 += " ";
let texte2 = "en force";
let texte3 = texte1 + texte2;

//Output : texte3 = "OFPPT en force"
```

L'application de l'opérateur + pour concaténer les chaînes de caractères et les nombres :

Exemple 2 :

```
let x = 1 + 1;
let y = "5" + 1;

//x=2
//y="51"
```

Opérateurs de comparaison en JavaScript

Les opérateurs de comparaison permettent de comparer des **opérandes** (qui peuvent être des valeurs numériques ou des chaînes de caractères) et renvoie une valeur logique : **true** (vrai) si la comparaison est vraie, **false** (faux) sinon.

Opérateur	Signification	Exemple
==	Egal à (comparaison des valeurs)	let x = 5; let y = "5"; let z=(x==y); //z=true
===	Egal à (comparaison de la valeur et du type)	let x = 5; let y = "5"; let z=(x===y); //z=false
!=	Différent de (n'est pas égal à)	let x = 5; let y = 5; let z=(x!=y); //z=false
!==	Type ou valeur différente	let x = 5; let y = "5"; let z=(x!==y); //z=true
>	Supérieur à	let x = 5; let y = 5; let z=(x>y); //z=false
<	Inférieur à	let x = 5; let y = 5; let z=(x<y); //z=false
>=	Supérieur ou égal à	let x = 5; let y = 5; let z=(x>=y); //z=true
<=	Inférieur ou égal à	let x = 5; let y = 5; let z=(x<=y); //z=true

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Expressions et opérateurs



Opérateurs logiques en JavaScript

Les opérateurs logiques, aussi appelés opérateurs booléens, sont utilisés pour combiner des valeurs booléennes (des conditions qui peuvent avoir les valeurs **true** ou **false**) et produire une nouvelle valeur booléenne.

Opérateur	Signification
&&	ET logique
	OU logique
!	NON logique

Opérateurs de type en JavaScript

Opérateur	Signification	Exemple
typeof	Retourne le type de la variable	typeof(5) retourne "number" typeof("5") retourne "string"
instanceof	Retourne true si l'objet est une instance de la classe donnée en paramètre	console.log("JavaScript" instanceof String);

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Expressions et opérateurs



Opérateurs bit-à-bit en JavaScript

Les opérateurs bit à bit sont des opérateurs qui permettent d'effectuer des transformations sur des nombres entiers de 32 bits comme des chiffres binaires.

Opérateur	Signification	Exemple	Équivalent à	Résultat binaire	Résultat décimal
&	ET binaire : Renvoie 1 pour chaque position de bits pour laquelle les bits correspondants des deux opérandes sont 1	5 & 1	0101 & 0001	0001	1
	OU binaire : Renvoie 1 pour chaque position de bits pour laquelle le bit correspondant d'au moins un des deux opérandes est 1	5 1	0101 0001	0101	5
~	NON binaire : inverse les bits de l'opérande	~ 5	~0101	1010	10
^	XOR binaire : Renvoie 1 pour chaque position de bit pour laquelle le bit correspondant d'un seul des deux opérandes est 1	5 ^ 1	0101 ^ 0001	0100	4
<<	Décalage de bits à gauche : $a \ll b$ décale "a" en représentation binaire de "b" bits vers la gauche, en introduisant des zéros par la droite	5 << 1	0101 << 1	1010	10
>>	Décalage de bits à droite: $a \gg b$ décale "a" en représentation binaire de "b" bits vers la droite, en rejetant les 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 permet d'afficher les données de différentes manières :

document.write() : Écriture dans le document de sortie HTML

- Utilisée pour des fins de test. Après le chargement du document HTML, elle supprime le contenu existant.

```
<!DOCTYPE html>
<html>
<body>
<h1>Page Web de test</h1>
<p>C'est un paragraphe</p>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

window.alert() : Écriture dans une boîte d'alerte

- Utilisée pour afficher des messages à l'utilisateur et aussi pour afficher des données

```
<!DOCTYPE html>
<html>
<body>
<h1> Page Web de test </h1>
<p> C'est un paragraphe </p>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

console.log() : Écriture dans la console du navigateur. Cette méthode est pratique pour le débogage du code

- Utilisée pour afficher des données dans la console du navigateur. Cette méthode est pratique pour le débogage du code.

```
<!DOCTYPE html>
<html>
<body>
<h1> Page Web de test </h1>
<p> C'est un paragraphe </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

La méthode **window.print()** permet d'imprimer le contenu de la fenêtre en cours en appelant le dispositif propre du navigateur.

L'exemple suivant permet d'appeler la méthode « window.print() » suite au clic sur un bouton.

Exemple :

```
<!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

En JavaScript, on peut récupérer les données de deux manières différentes :

Prompt : affiche une boîte de dialogue avec une zone de saisie

- La méthode **prompt** (boite d'invite) propose un champ comportant une entrée à compléter par l'utilisateur avec une valeur par défaut.
- En cliquant sur OK, la méthode renvoie la valeur tapée par l'utilisateur ou la réponse proposée par défaut. Si l'utilisateur clique sur Annuler (Cancel), la valeur **null** est alors renvoyée.

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

Les formulaires : les contrôles des formulaires comme la zone <input>

- Javascript peut récupérer les données de n'importe quel élément html par la méthode : **document.getElementById(id)** qui prend en paramètre l'identifiant de l'objet.

```
<!DOCTYPE html>  
<html>  
<body>  
<input type="text" id="prenom">  
<button  
onlick="alert(document.getElementById('prenom')  
.value)">Afficher</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, on distingue deux familles de types de variables :

Types primitifs

- **String** : chaînes de caractères
- **Number** : nombres entiers et réels
- **Boolean** : les booléens (true, false)
- **Undefined** : valeur prise par les variables déclarées non initialisées

Les types structurels

- **Date** : pour représenter les dates
- **Array** : pour représenter les tableaux
- **Object** : tel que les enregistrements
- **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



Données primitives

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

L'opérateur **typeof** permet de renvoyer les types primitifs suivants :

- string
- number
- boolean
- undefined

Exemple :

```
typeof "Hassan"           // Retourne "string"  
typeof 3.14               // Retourne "number"  
typeof NaN               // 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** permet aussi de renvoyer les types complexes suivants :

- **Function** : pour les fonctions
- **Object** : renvoyer les objets, les tableaux et null.

Exemple :

```
typeof {name:'Hassan', age:34} // Retourne "object"  
typeof [1,2,3,4]                // Retourne "object"  
typeof null                    // Retourne "object"  
typeof function myFunc(){}
```

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Types primitifs et objets de base



Conversion de type en JavaScript

On peut convertir le type des variables JavaScript :

- Utilisant les fonctions JavaScript proposées ;
- Automatiquement par JavaScript lui-même.

Conversion de chaînes en nombres

La méthode **Number()** permet de convertir les chaînes en nombres selon les règles suivantes :

- Les chaînes de caractères 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 (Not a Number).

Exemple :

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

Si la variable de type chaîne ne peut pas être convertie, elle prend la valeur **NaN**.

Conversion de nombres en chaînes

La méthode **String()** est utilisée pour convertir des nombres en chaînes de caractères. Elle s'applique sur tout type de nombres, les littéraux, et les expressions :

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

La méthode **toString()** est équivalente :

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

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Types primitifs et objets de base



Conversion des dates en nombres

La méthode **Number()** peut être utilisée pour convertir des dates en nombres :

```
d = new Date(); // retourne object  
Number(d)      // retourne number
```

La méthode de date **getTime()** est équivalente :

```
d = new Date(); // retourne Object  
d.getTime();   // retourne Number
```

Conversion de dates en chaînes

La méthode **String()** permet de convertir des dates en chaînes de caractères :

```
String(Date()) // retourne une chaîne représentant la date actuelle complète
```

La méthode de date **toString()** est équivalente :

```
Date().toString() // retourne une chaîne représentant la date actuelle complète
```

01 - Maîtriser la syntaxe javascript et ses notions fondamentales

Types primitifs et objets de base



Conversion de dates en chaînes

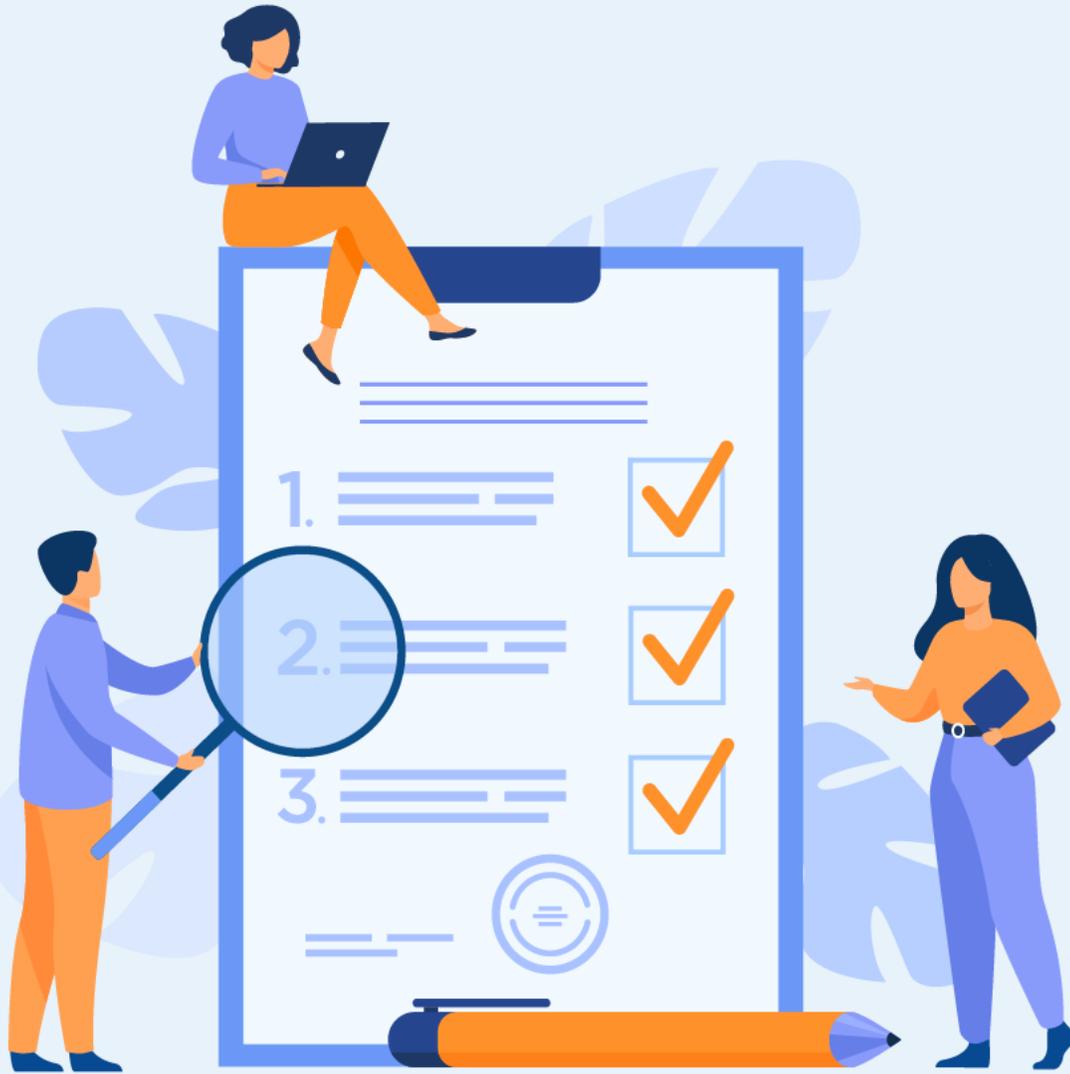
Méthode	Description
getDate()	Renvoie le numéro du jour (1-31)
getDay()	Renvoie le numéro du jour de la semaine (0-6)
getFullYear()	Renvoie l'année en 4 chiffres (yyyy)
getHours()	Renvoie l'heure (0-23)
getMilliseconds()	Renvoie les millisecondes (0-999)
getMinutes()	Renvoie les minutes (0-59)
getMonth()	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

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

Syntaxe :

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

Exemple :

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

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

02 - Maîtriser les structures de contrôle

Structures alternatives



La déclaration else

L'instruction **else** est utilisée pour spécifier un bloc de code à exécuter si la condition est fausse.

Syntaxe :

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

L'instruction **else if** est utilisée 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 condition2 est vraie et la condition1 est fausse  
} else {  
    // bloc d'instructions à executer si la condition2 est fausse  
}
```

Exemple :

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

L'instruction switch

L'instruction **switch** est utilisée pour sélectionner un choix parmi plusieurs (pour remplacer les blocs if .. else multiples).

Syntaxe

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

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";  
}
```

- L'instruction **switch** évalue une expression et fait correspondre la valeur de l'expression à un des cas déclarés. Elle exécute les instructions associées à ce cas (**case**), ainsi que les instructions des cas suivants.
- S'il n'y a pas de correspondance, le bloc de code par défaut est exécuté.
- Lorsque JavaScript atteint le mot clé **break**, il sort du bloc switch.

CHAPITRE 2

Maîtriser les structures de contrôle

1. Structures alternatives
2. **Structures itératives**



02 - Maîtriser les structures de contrôle

Structures itératives

Les types de boucles

JavaScript, comme la plupart des langages de programmation, prend en charge les types de boucles suivantes :



02 - Maîtriser les structures de contrôle

Structures itératives



La boucle For

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

Array.forEach()

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

Array.forEach()

```
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** est utilisée pour sortir d'une boucle.

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

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

La déclaration continue

L'instruction **continue** ignore une itération (dans la boucle), si une condition spécifiée est vraie, et passe à l'itération suivante dans la boucle.

Dans l'exemple suivant, la valeur 3 est ignorée :

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

CHAPITRE 3

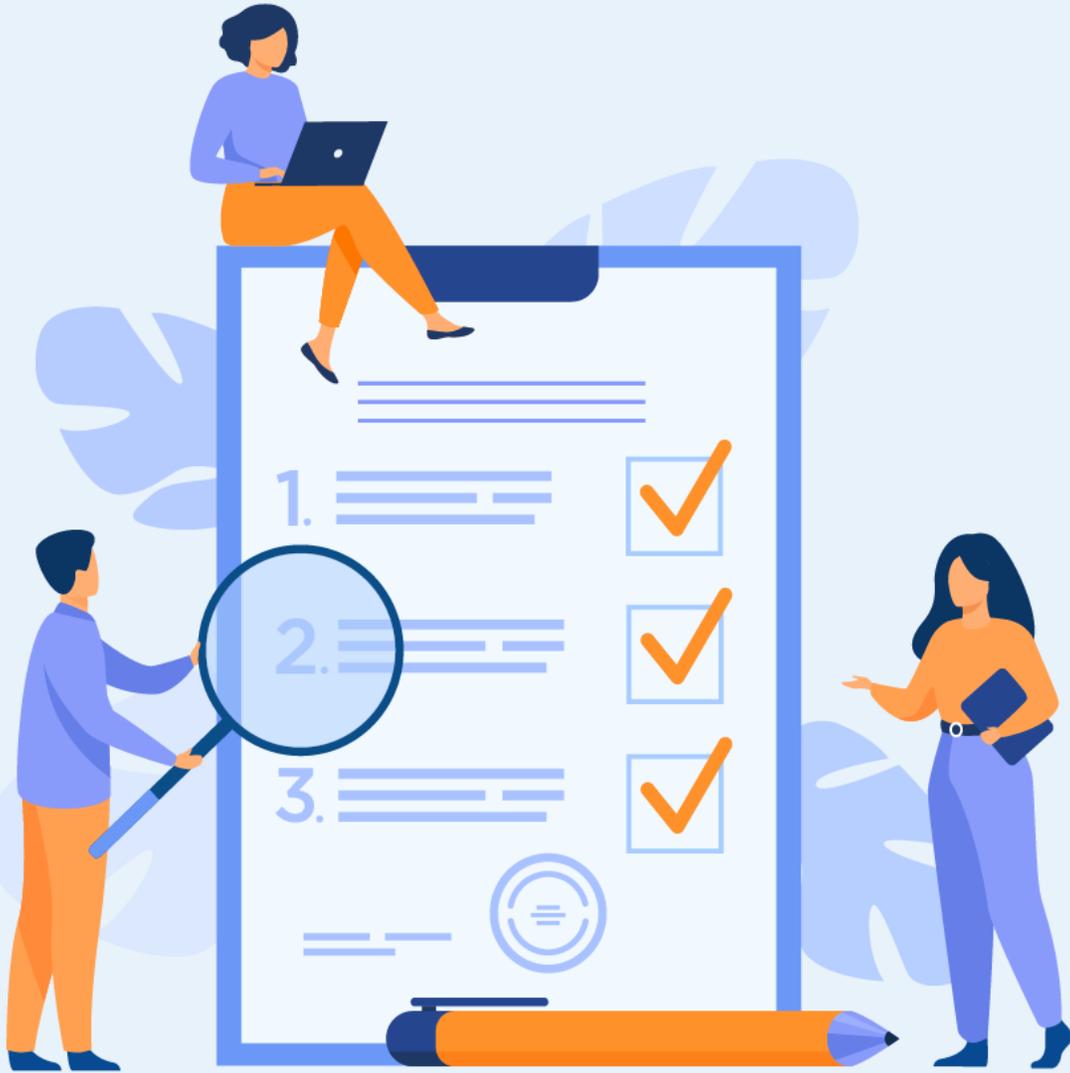
Utiliser des fonctions

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



Définition

- Une fonction est définie comme un bloc de code organisé et réutilisable, créé pour effectuer une action unique ;
- Le rôle des fonctions est d'offrir une meilleure modularité au code et un haut degré de réutilisation ;
- Une fonction JavaScript est un ensemble d'instructions qui peut prendre des entrées de l'utilisateur, effectuer un traitement et renvoyer le résultat à l'utilisateur ;
- Une fonction JavaScript est exécutée au moment de son appel.

En JavaScript, il existe 2 types de fonctions :

Les fonctions propres à JavaScript : appelées "méthodes". Elles sont associées à un objet bien particulier comme la méthode `alert()` avec l'objet `window`.

Les fonctions écrites par le développeur : déclarations placées dans l'en-tête de la page.

Syntaxe des fonctions en JavaScript

- En JavaScript, une fonction est définie avec le mot clé **function**, suivi du nom de la fonction , et des parenthèses () ;
- Le nom de la fonction doit respecter les mêmes règles que les noms des variables ;
- Les parenthèses peuvent inclure des noms de paramètres séparés par des virgules. Les arguments de la fonction correspondent aux valeurs reçues par la fonction lorsqu'elle est invoquée ;
- Le code à exécuter, par la fonction, est placé entre accolades : {}

```
function nomFonction(paramètre1, paramètre2, paramètre3, ...)  
{  
    // Code de la fonction  
    return ... ;  
}
```

Retour de la fonction

- L'instruction **return** est utilisée pour renvoyer une valeur (souvent calculée) au programme appelant.
- Lorsque l'instruction **return** est atteinte, la fonction arrête son exécution.

Appel de fonction

Le code de la fonction est exécuté quand la fonction est appelée selon les cas suivants :

- Lorsqu'un événement se produit (clic sur un bouton par exemple) ;
- Lorsqu'il est invoqué (appelé) à partir d'un autre code JavaScript ;
- Automatiquement (auto-invoqué).

La fonction est appelée en utilisant son nom et, si elle prend des paramètres, en indiquant la liste de ses arguments :

```
nomFonction(p1, p2, p3, ...)
```

Exemple : fonction avec retour

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

Variables locales à la fonction

- A l'intérieur de la fonction, les arguments (les paramètres) sont considérés comme des variables locales.
- 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"
```



Remarques

- 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



Les expressions lambdas

- Les fonctions fléchées (arrow functions) sont des fonctions qui ont une syntaxe compacte. Par conséquent, elles sont plus rapide à écrire que les fonctions traditionnelles ;
- Les fonctions fléchées sont limitées et ne peuvent pas être utilisées dans toutes les situations ;
- Principe: à la place du mot clé function, on utilise le signe (=>) plus une parenthèse carrée fermante (>) après la parenthèse fermante de la fonction :

Exemple 1 :

```
const variable = () => {  
  return "ma_variable"  
}  
console.log(variable()) // "ma_variable"
```

Exemple 2 :

```
const variable = (a,b) => {  
  return a*b;  
}  
console.log(variable(2,3)) // 6
```

03 - Utiliser des fonctions

Expressions lambdas



Les expressions lambdas

Exemple 3 :

```
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 définie par la fonction: (L)=>L.length.

Exemple 4 :

```
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)

- Les fonctions JavaScript sont exécutées dans l'ordre où elles sont appelées. Pas dans l'ordre où elles sont définies ;
- Une fonction peut appeler une fonction dans son code, directement ou en utilisant les fonctions de rappel ;
- Une fonction de rappel (**callback** en anglais) est une fonction passée en paramètre d'une autre fonction en tant qu'argument ;
- La fonction de rappel est invoquée à l'intérieur de la fonction externe pour exécuter des instructions précises.

Exemple : [https://www.w3schools.com/js/js_callback.asp]

```
function affichage(s) {  
  console.log(s);  
}  
  
function calcul(num1, num2, myCallback) {  
  let somme = num1 + num2;  
  myCallback(somme);  
}  
  
calcul(1, 5, affichage);
```

Dans l'exemple, « affichage » est le nom d'une fonction qui est passée à la fonction calcul() comme argument.

Attention : ne pas utiliser les parenthèses dans l'appel de la fonction.

03 - Utiliser des fonctions

Appels asynchrones (callback, Promise)



Les promesses (Promise)

- Une promesse (**Promise**) est un objet JavaScript qui contient à la fois le code producteur et les appels au code consommateur : renvoie une valeur qu'on va utiliser dans le futur.
- La promesse est adaptée à la gestion des opérations asynchrones.

Exemple :

```
Instruction1  
Instruction2  
Instruction3 //Qui récupère une valeur externe et la mettre dans un  
élément HTML (cela prend 5 secondes)  
Instruction4
```

Problème : Le code actuel provoque une attente de 5 secondes avant d'exécuter l'instruction4.

Solution : en utilisant un objet Promise, on peut provoquer l'exécution de l'instruction3 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 (Promise)

En JavaScript, une promesse a trois états :

- **En attente (Pending)** : résultat indéfini ;
- **Accompli (Fulfilled)** : opération réussie, le résultat est une valeur ;
- **Rejeté (Rejected)** : le résultat est une erreur.

Une promesse commence toujours dans l'état « en attente » et se termine par un des états « accompli » ou « rejeté » comme illustré sur la figure suivante :

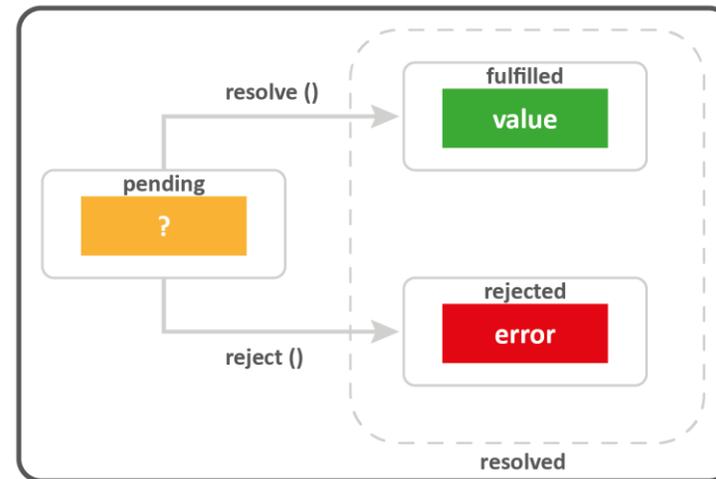


Figure 7 : Les états d'une promesse

03 - Utiliser des fonctions

Appels asynchrones (callback, Promise)



Créer une promesse : le constructeur Promise

Pour créer une promesse en JavaScript, on utilise le constructeur **Promise** :

Exemple :

```
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 l'exécuteur (**executor**).
- L'exécuteur accepte deux fonctions avec les noms, par convention, **resolve()** et **reject()**.
- À l'intérieur de l'exécuteur, on appelle manuellement la fonction **resolve()** si l'exécuteur est terminé avec succès et la fonction **reject()** en cas d'erreur.

03 - Utiliser des fonctions

Appels asynchrones (callback, Promise)



Créer une promesse : le constructeur Promise

L'exécution du code suivant montre que la promesse est résolue car la variable **completed** a la valeur initiale true. Pour voir l'état d'attente de la promesse, on utilise la fonction **setTimeout()** :

Exemple 3 :

```
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);
});
```

Résultat d'exécution : la promesse commence par l'état **pending** avec la valeur **undefined**. La valeur promise sera renvoyée ultérieurement une fois la promesse terminée.

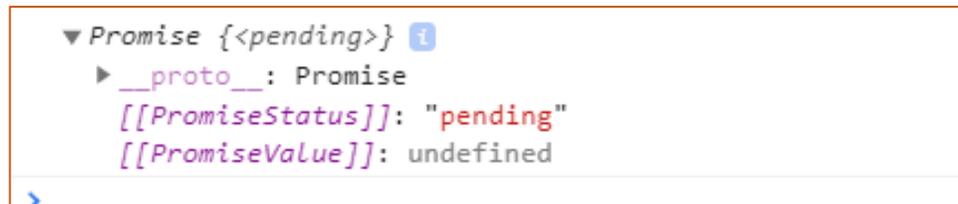


Figure 8 : Exemple d'exécution d'une promesse Javascript (étape 1)

03 - Utiliser des fonctions

Appels asynchrones (callback, Promise)

Créer une promesse : le constructeur Promise

Après le passage de 3 secondes, l'affichage de la variable `getData` montre que l'état de la promesse devient **resolved** et la valeur de la promesse est la chaîne 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

Figure 9 : Exemple d'exécution d'une promesse Javascript (étape 2)

L'appel de la fonction `resolve()` bascule l'état de l'objet de promesse vers l'état **fulfilled**. Si on affecte `false` à la variable `completed` on aura le résultat suivant :

```
✖ 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"
```

Figure 10 : Exemple d'exécution d'une promesse Javascript (étape 3)

03 - Utiliser des fonctions

Appels asynchrones (callback, Promise)



Consommer une promesse : then, catch, finally

Les fonctions `then()`, `catch()`, et `finally()` sont utilisées pour planifier un rappel lorsque la promesse finit dans les états « résolue » ou « rejetée ».

1. La méthode `then()`

Cette méthode est utilisée pour planifier l'exécution d'un rappel. Elle peut prendre deux fonctions de rappel :

Le **callback `onFulfilled`** (appelé si la promesse est accomplie) et le **callback `onRejected`** (appelé lorsque la promesse est rejetée) :

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

Exemple : 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);  
  });  
}
```

Appel de la fonction `makePromise()` et invocation de la méthode `then()` :

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

03 - Utiliser des fonctions

Appels asynchrones (callback, Promise)



Consommer une promesse : then, catch, finally

2. La méthode catch()

La méthode `catch()` est utilisée pour programmer un rappel à exécuter lorsque la promesse est rejetée. En interne, la méthode `catch()` appelle la méthode `then(undefined, onRejected)`.

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

3. La méthode finally()

La méthode `finally()` est utilisée pour exécuter un code quelque soit l'état de la promesse :

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

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



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

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

Lorsqu'une erreur se produit, JavaScript arrête et crée un **objet d'erreur** (error) avec deux propriétés : **nom** et un **message** (varie selon le navigateur) → JavaScript lance **une exception**, on dit qu'une erreur est générée.

Pour gérer les exceptions en JavaScript on utilise les mots clés suivants :

- **try** : tester un bloc de code pour les erreurs ;
- **catch** : gérer l'erreur ;
- **throw** : créer des erreurs personnalisées ;
- **Finally** : permet l'exécution de code, après try et catch, quel que soit le résultat.

Syntaxe :

```
try {  
    //Bloc du code à exécuter  
}  
catch(err) {  
    //Bloc du code qui gère l'exception  
}
```

03 - Utiliser des fonctions

Gestion des exceptions



Le bloc try ... catch

- L'instruction throw

L'instruction throw permet de créer une erreur personnalisée.

L'exception à générer peut être de type String, Number, un Boolean ou un Object

Exemple :

```
if(a > 200)
    throw "Too big";    // exception de type string
else
    throw 500;         // exception de type number
```

L'utilisation de l'instruction throw avec try et catch permet de contrôler le déroulement du programme et de générer des messages d'erreurs personnalisés.

03 - Utiliser des fonctions

Gestion des exceptions



Exemple de validation d'entrée

Dans l'exemple suivant [], l'utilisateur doit saisir une valeur comprise entre 1 et 5.

On distingue 4 cas :

- **x est vide** : exception levée avec le message « **Vide** » ;
- **x n'est pas un nombre** : exception levée avec le message « **Ce n'est pas un numéro** » ;
- **x < 5** : exception levée avec le message « **Trop petit** » ;
- **x > 10** : exception levée avec le message « **Trop grand** ».

```
let x = Number(prompt("Donnez un numéro entre 5 et 10"));
try {
  if(x == "") throw "Vide";
  if(isNaN(x)) throw "Ce n'est pas un numéro";
  x = Number(x);
  if(x < 5) throw "Trop petit";
  if(x > 10) throw "Trop grand";
}
catch(err)
{
  console.log(err);
}
```

03 - Utiliser des fonctions

Gestion des exceptions



L'objet error

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

Valeurs de nom d'erreur :

Nom de l'erreur	Description
EvalError	Erreur dans la fonction eval()
RangeError	Nombre hors limite
ReferenceError	Référence inconnue
SyntaxError	Erreur de syntaxe
TypeError	Une erreur de type s'est produite
URIError	URI incorrecte

Exemple :

```
let x = 5;
try {
  x = y + 1; // y n'est pas référencé en mémoire
}
catch(err) {
  console.log(err.name);
}
```

Output : ReferenceError

CHAPITRE 4

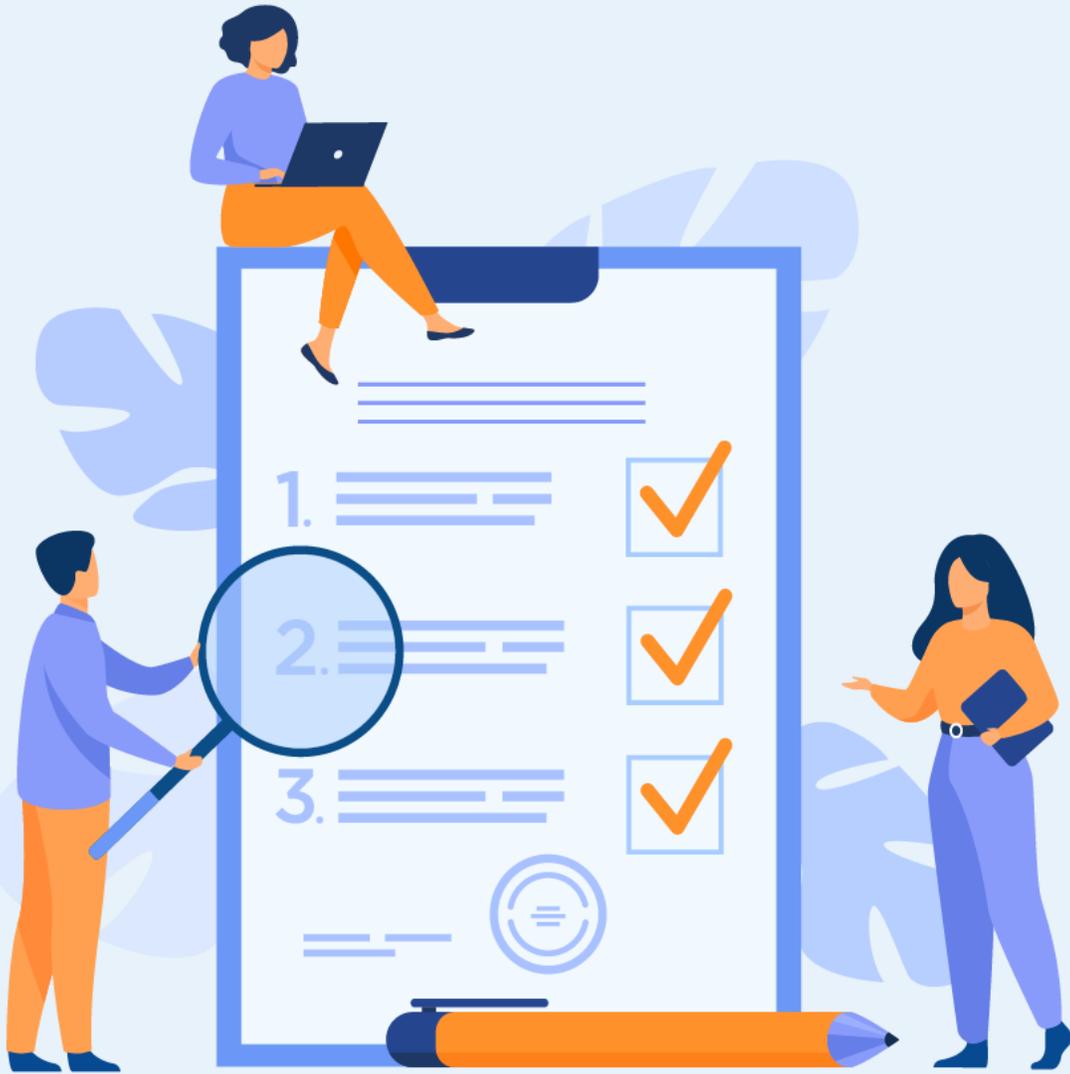
Manipuler les objets

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



Création d'objet avec syntaxe littérale

La syntaxe littérale consiste à créer un objet en définissant ses propriétés à l'intérieur d'une paire d'accolades : { ... } :

Syntaxe :

```
const monObjet = {  
  propriete1: valeur1,  
  propriete2: valeur2,  
  // ... ,  
  methode1(/* ... */) {... // ..},  
  methode2(/* ... */) {... // ...}  
// ...  
};
```



Remarques

- Une méthode est une propriété dont la valeur est une fonction. Son rôle est de définir un comportement (action) pour l'objet.
- On peut utiliser var au lieu de const.

Exemple :

```
const telephone = {  
  marque: 'SmartF22',  
  prix: 400,  
  stock: 200,  
  ref: 'SmartPhone2022',  
  VerifierStock: function() {  
    if (this.stock > 0) {return true;}  
    else { return false;}  
  }  
}
```

Test :

```
console.log(telephone.marque); //SmartF22  
console.log(telephone.VerifierStock()); //True
```

Dans l'exemple, l'objet « telephone » est caractérisé par les propriétés : marque, prix, stock et ref. Il possède aussi la méthode VerifierStock() qui retourne True si le stock est disponible (>0).

Création d'objet avec constructeur

La création des objets en utilisant un constructeur permet d'instancier cet objet dans des variables différentes :

Syntaxe :

```
function monObjet(param1, param2, ...){  
    this.propriete1: param1,  
    this.propriete2: param2,  
    // ... ,  
    this.methode1 = function ()  
        {... // ..},  
    // ...  
};
```

Exemple :

```
function Telephone(n, p, s, r) {  
    this.nom = n;  
    this.prix = p;  
    this.stock = s;  
    this.ref = r;  
    this.VerifierStock = function() {  
        if (this.stock > 0) {return true;}  
        else {return false;}  
    }  
}
```

Test :

```
var t1 = new Telephone("SmartF22", 400, 200, "pro1");  
var t2= new Telephone("t2", 200, 0, "Mi Max");  
console.log(t1.nom); //SmartF22  
console.log(t2.VerifierStock()); //False
```

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 d'objet



Ajouter/modifier des propriétés ou des méthodes

On peut ajouter des méthodes et des propriétés aux objets créés : `telephone.dateSortie= '2022'`;

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

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

Output :

Marque

Prix

Stock

Ref

VerifierStock

dateSortie

Supprimer une propriété ou une fonction

On supprime une propriété en utilisant le mot clé `delete` : `delete telephone.dateSortie;`

CHAPITRE 4

Manipuler les objets

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



Les tableaux : déclaration

Un tableau JS est un objet qui hérite de l'objet global standard **Array** (héritant de **Object**).

L'objet **Array** est une liste d'éléments indexés dans lesquels on pourra ranger (écrire) et reprendre (lire) des données.

- **Déclaration d'un tableau vide :**

```
let tableau = new Array; // déclaration explicite en instanciant l'objet Array
let tableau = new Array(3); // 3 correspond à la taille du tableau

let tableau = []; // déclaration de manière littérale avec []
```

- **Déclaration et initialisation :**

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

- **Taille du tableau :** La propriété **length** de l'objet **Array** retourne la taille du tableau :

```
let tableau = ['A', 'B', 'C'];
let nbElements = tableau.length;
console.log(nbElements); // 3
```

04 - Manipuler les objets

Manipulation des objets natifs



Accéder aux éléments du tableau

Pour accéder à un élément du tableau, on utilise son indice : `nom_du_tableau[i] = "élément";`

Rappel : Les indices du tableau commencent à 0

- Accéder à un élément du tableau :

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

- Récupérer l'indice d'un élément : la méthode `indexOf()` :

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

- `indexOf()` a un deuxième paramètre optionnel qui permet aussi de choisir l'indice à partir duquel la recherche débute (Par défaut, ce deuxième paramètre est à 0) :

```
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



Parcourir un tableau

On peut parcourir un tableau de différentes manières :

```
for (let i = 0; i < monTableau.length; i++)  
{  
  // monTableau[i] permet d'accéder à l'élément courant du tableau  
}  
//Ou bien  
for (const monElement of monTableau)  
{  
  // monElement permet d'accéder à l'élément courant du tableau  
}
```

```
monTableau.forEach ( monElement => {  
  // monElement permet d'accéder à l'élément courant du tableau  
});  
//Ou bien  
monTableau.forEach ( function ( monElement )  
{  
  // monElement permet d'accéder à l'élément courant du tableau  
});
```

Exemple :

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

04 - Manipuler les objets

Manipulation des objets natifs



Ajouter un élément dans un tableau

- La méthode **push** ajoute un élément à la fin du tableau :

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

- La méthode **unshift** ajoute un élément au début du tableau :

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

Modifier un élément du tableau

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

```
let tableau = ['B', 'B', 'C'];  
tableau[0] = 'A'; // Modifie la première case  
tableau[4] = 'E'; // ajoute un élément dans l'indice 4  
console.log(tableau); // Retourne ['A', 'B', 'C', empty, 'E']
```

Supprimer un élément du tableau

- La méthode **pop()** supprime le dernier élément du tableau :

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

- La méthode **shift()** supprime le premier élément du tableau :

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

- La méthode **splice()** permet de supprimer plusieurs éléments :

Le premier paramètre est l'indice à partir duquel supprimer, et le second est le nombre d'éléments à supprimer.

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

04 - Manipuler les objets

Manipulation des objets natifs



Trier un tableau

- La méthode `sort()` retourne les éléments par ordre alphabétique (elle doivent être de la même nature) :

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

- La méthode `reverse()` Inverse l'ordre des éléments (sans tri) :

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

Recherche un élément dans le tableau

- La méthode `findIndex()` retourne l'indice du premier élément du tableau qui remplit une condition :

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

L'objet String (chaînes de caractères)

L'objet **String** de JavaScript est utilisé pour manipuler les chaînes de caractères.

Il possède de nombreuses méthodes :

Méthode	Description
length	C'est un entier qui indique la taille de la chaîne de caractères.
charAt()	Méthode qui permet d'accéder à un caractère isolé d'une chaîne.
substring(x,y)	Méthode qui renvoie un string partiel situé entre la position x et la position y-1.
toLowerCase()	Transforme toutes les lettres en minuscules.
toUpperCase()	Transforme toutes les lettres en Majuscules.

- Chercher dans une Chaîne : **indexOf**, **startsWith**, **endsWith**, **split**, ...
- Transformer une chaîne en tableau : la méthode **Array.from()** permet de transformer une chaîne en un véritable tableau.

04 - Manipuler les objets

Manipulation des objets natifs



L'objet Date

L'objet **date** de JavaScript est utilisé pour obtenir la date (année, mois et jour) ainsi que le temps (heure, minutes et secondes).

On peut 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 l'objet Date JavaScript (1)

Méthodes	Description
<code>getDate()</code>	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>	Renvoie la valeur entière comprise entre 0 et 6 qui représente le jour de la semaine sur la base de l'heure locale.
<code>getFullYears()</code>	Renvoie la valeur entière qui représente l'année sur la base de l'heure locale.
<code>getHours()</code>	Renvoie la valeur entière comprise entre 0 et 23 qui représente les heures sur la base de l'heure locale.
<code>getMilliseconds()</code>	Renvoie la valeur entière comprise entre 0 et 999 qui représente les millisecondes sur la base de l'heure locale.
<code>getMinutes()</code>	Renvoie la valeur entière comprise entre 0 et 59 qui représente les minutes sur la base de l'heure locale.
<code>getMonth()</code>	Renvoie la valeur entière comprise entre 0 et 11 qui représente le mois sur la base de l'heure locale.
<code>getSeconds()</code>	Renvoie la valeur entière comprise entre 0 et 60 qui représente les secondes sur la base de l'heure locale.
<code>getUTCDate()</code>	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>	Renvoie la valeur entière comprise entre 0 et 6 qui représente le jour de la semaine sur la base de l'heure universelle.
<code>getUTCFullYears()</code>	Renvoie la valeur entière qui représente l'année sur la base du temps universel.
<code>getUTCHours()</code>	Renvoie la valeur entière comprise entre 0 et 23 qui représente les heures sur la base du temps universel.
<code>getUTCMinutes()</code>	Renvoie la valeur entière comprise entre 0 et 59 qui représente les minutes sur la base du temps universel.
<code>getUTCMonth()</code>	Renvoie la valeur entière comprise entre 0 et 11 qui représente le mois sur la base du temps universel.
<code>getUTCSeconds()</code>	Renvoie la valeur entière comprise entre 0 et 60 qui représente les secondes sur la base du temps universel.
<code>setDate()</code>	Définit la valeur du jour pour la date spécifiée sur la base de l'heure locale.
<code>setDay()</code>	Définit le jour particulier de la semaine sur la base de l'heure locale.
<code>setFullYears()</code>	Définit la valeur de l'année pour la date spécifiée sur la base de l'heure locale.

Méthodes de l'objet Date JavaScript (2)

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

04 - Manipuler les objets

Manipulation des objets natifs



L'objet Math

L'objet **Math** de JavaScript fournit un ensemble de constantes et méthodes pour effectuer des opérations mathématiques. Contrairement à l'objet `date`, il n'a pas de constructeurs.

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

L'objet Window

L'objet **window** de JavaScript est le parent de chaque objet qui compose la page web. Il possède plusieurs méthodes :

La méthode **alert** : bloque le programme tant que l'utilisateur n'aura pas cliqué sur "OK". Utile pour le déboguer les scripts.

Syntaxe :

```
alert(variable) ;  
alert("chaîne de caractères") ;  
alert(variable+ "chaîne de caractères");
```

La méthode **confirm(texte)** : permet d'avertir l'utilisateur en ouvrant une boîte de dialogue avec deux choix "OK" et "Annuler ". Le clic sur OK renvoie la valeur true.

Syntaxe :

```
if (confirm('texte'))  
{  
    //action à faire pour la valeur true    }  
else  
{  
    //action à faire pour la valeur false    }
```

La méthode **prompt (boîte d'invite)** propose un champ comportant une entrée à compléter par l'utilisateur. Cette entrée possède aussi une valeur par défaut. En cliquant sur OK, la méthode renvoie la valeur tapée par l'utilisateur ou la réponse proposée par défaut. Si l'utilisateur clique sur Annuler ou Cancel, la valeur null est alors renvoyée.

Syntaxe :

```
prompt("texte de la boîte d'invite" , "valeur par défaut") ;
```

04 - Manipuler les objets

Manipulation des objets natifs



L'objet Window

La méthode open permet d'ouvrir une nouvelle fenêtre.

Syntaxe :

```
[window.]open("URL","nom_de_la_fenêtre","caractéristiques_de_la_fenêtre")
```

- URL est l'URL de la page que l'on désire afficher dans la nouvelle fenêtre.
- Caractéristiques_de_la_fenêtre est une liste de certaines ou de toutes les caractéristiques de la fenêtre.

Quelques caractéristiques :

- **height=pixels** : la hauteur de la fenêtre (valeur minimale est 100 px) ;
- **width=pixels** : la largeur de la fenêtre (valeur minimale est 100 px) ;
- **left=pixels** : la position de la fenêtre à partir de la gauche.

04 - Manipuler les objets

Manipulation des objets natifs



L'objet Window

Les méthodes `setTimeout()` et `clearTimeout()` permettent de déclencher une fonction après un laps de temps déterminé.

Syntaxe :

```
nom_du_compteur = setTimeout("fonction_appelée()", temps en milliseconde)
```

Exemple : lancer la fonction `démarrer()` après 5 secondes.

```
setTimeout("démarrer()",5000)
```

Pour arrêter le temporisateur avant l'expiration du délai fixé :

```
clearTimeout(nom_du_compteur) ;
```

04 - Manipuler les objets

Manipulation des objets natifs



L'objet Window

La méthode **setInterval ()** appelle une fonction ou évalue une expression à des intervalles spécifiés (en millisecondes).

La méthode **setInterval ()** continue d'appeler la fonction jusqu'à ce que la méthode **clearInterval ()** soit appelée ou que la fenêtre soit fermée.

Syntaxe :

```
var x = setInterval(fonction, temps)  
...  
clearInterval(x)
```

Les expressions régulières (regex)

- Les expressions régulières (**Regular Expressions - Regex**) sont des patrons (exprimés sous forme de combinaisons de caractères) permettant d'effectuer des opérations de recherche et de remplacement sur un texte.
- En JavaScript, les expressions régulières sont des objets (**RegExp Object**) possédant des propriétés et des méthodes.

Syntaxe :

Pour créer une expression régulière en JavaScript, il faut entourer le patron (pattern) par les caractères (/):

```
let Expr = /ofppt/;
```

Ou bien en utilisant le constructeur **RegExp** de JavaScript :

```
let Expr = new RegExp('ofppt');
```

Les expressions régulières (regex)

Les méthodes utilisées dans les expressions régulières sont listées dans le tableau ci-dessous :

Méthode	Description
exec()	Cherche une correspondance (match) d'un pattern dans une chaîne de caractères. Retourne un tableau ou null.
test()	Cherche une correspondance (match) d'un pattern dans une chaîne de caractères. Retourne true ou false.
match()	Retourne null ou un tableau contenant toutes les correspondances.
matchAll()	Retourne un itérateur contenant toutes les correspondances.
search()	Teste une correspondance dans une chaîne de caractères. Retourne -1 ou l'index de la correspondance.
replace()	Cherche une correspondance dans une chaîne et la remplace par une sous-chaîne.
replaceAll()	Recherche toutes les correspondances dans une chaîne et les remplace par une sous-chaînes.
split()	Décompose une chaîne en un tableau de sous-chaînes selon une expression régulière.

Modificateurs d'expressions régulières

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

Modificateur
i
g
m

Quantificateurs d'expressions régulières

Les quantificateurs définissent les quantités :

Expression	Description
n+	Correspond à toute chaîne contenant au moins un caractère n.
n*	Correspond à toute chaîne contenant zéro ou plusieurs occurrences du caractère n.
n?	Correspond à toute chaîne contenant zéro ou une occurrence du caractère n.

04 - Manipuler les objets

Manipulation des objets natifs



Modèles d'expressions régulières

Les crochets sont utilisés pour rechercher une plage de caractères :

Modificateur	Description
[abc]	Trouver l'un des caractères entre les crochets → a ou b ou c
[0-9]	Trouver l'un des chiffres entre les crochets → 0, 1, 2, ... ou 9
(x y)	Trouvez l'une des alternatives séparées par → x ou y

Les métacaractères

Les métacaractères sont des caractères ayant une signification particulière :

Expression	Description
\d	Trouver un chiffre
\s	Trouver un caractère d'espacement
\b	Trouver une correspondance au début ou à la fin d'un mot : \bMOT ou MOT\b
\uxxxx	Trouver le caractère Unicode spécifié par le nombre hexadécimal xxxx

Les assertions dans les expressions régulières

Les assertions sont utilisées pour indiquer les débuts et les fins des lignes et des mots :

Caractère	Description
<code>^</code>	Correspond au début de la chaîne
<code>\$</code>	Correspond à la fin de la chaîne
<code>\b</code>	Délimite un mot : la position où un caractère de mot n'est pas suivi ou précédé d'un autre caractère comme entre une lettre et un espace.

04 - Manipuler les objets

Manipulation des objets natifs



Les regex - Utilisation des méthodes search() et replace()

Exemple 1 :

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

```
let texte = "Cours JS";  
let pos = texte.search("JS");  
console.log(pos); //renvoie 6
```

Exemple 3 :

Utilisez une expression régulière insensible à la casse pour remplacer « JS » par « JavaScript » dans une chaîne :

```
let texte = "Un cours de JS";  
let NvTexte = texte.replace(/js/i, "JavaScript");  
console.log(NvTexte); //Un cours de JavaScript
```

Affiche le même résultat pour la chaîne « Un cours de js »

Exemple 2 :

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

```
let texte = "Cours JS";  
let NvTexte = texte.replace("JS", "JavaScript");  
console.log(NvTexte); //cours JavaScript
```

Exemple 4 :

Utilisez une expression régulière pour effectuer une recherche dans une chaîne :

```
let Expr = /[A-Z]/;  
let texte = "un cours de JavaScript";  
let index = texte.search(Expr);  
console.log(index); //12, indice de « J »
```

Affiche « -1 » si Expr=/[0-9]/

04 - Manipuler les objets

Manipulation des objets natifs



Les regex - Utilisation des méthodes test(), exec() et match()

Exemple 1 :

Tester si une chaîne contient une sous-chaîne :

```
let texte = "Cours JS";  
let p = texte.test("JS");  
console.log(p); //renvoie True
```

Ce code affiche « False » pour test("js") ;

Exemple 3 :

Utilisation de la fonction match()

```
let regex = /cours(=? js)/g;  
console.log('cours js'.match(regex)); // [ 'cours' ]  
console.log('cours html'.match(regex)); // null  
console.log('C\'est le cours js pour vous'.match(regex)); // [ 'cours' ]  
console.log('C\'est le premier cours du mois.'.match(regex)); // null
```

Exemple 2 :

Ce code affiche correct si le numéro de téléphone est écrit sous la forme ###-###-####

```
let tel = /^(?:\d{3}|\(\d{3}\))([\d-  
\.])\d{3}\1\d{4}$/; var OK = tel.exec("039-494-  
9499");  
if (!OK)  
    console.log('incorrect');  
else  
    console.log('ok');
```

CHAPITRE 4

Manipuler les objets

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



JSON : définition et caractéristiques

JSON (JavaScript Object Notation) est un format d'échange de données qui est facile à utiliser par les humains et les machines. Ce format est utilisé pour échanger les valeurs entre les applications (clients) et les serveurs. Sa forme complète est en notation d'objet JavaScript.

Caractéristiques de JSON :

- **Facile à utiliser** - L'API JSON offre une mise en œuvre avancée des différents types et structures de données, ce qui aide à simplifier les cas d'utilisation.
- **Performance et rapidité** - JSON consomme très peu de mémoire.
- **Outil gratuit** - la bibliothèque JSON est open source et gratuite.
- **Dépendance** - la bibliothèque JSON ne nécessite pas l'utilisation d'une autre bibliothèque pour le traitement.
- **Compatibilité :**
 - JSON est supporté par les navigateurs ;
 - JSON est pris en charge par tous les principaux framework JavaScript ;
 - JSON permet de transmettre et de sérialiser des données structurées à l'aide d'une connexion réseau ;
 - JSON est compatible avec des langages de programmation modernes.

JSON : Syntaxe

Règles générales de la syntaxe JSON

- Les données Json sont écrites entre accolades (braces) ;
- Les données sont représentées sous forme de paires de clé – valeur ;
- Les clés doivent être mises entre guillemets (double quotes) ;
- La clé et la valeur doivent être séparées par deux points (:)
- La virgule (,) est utilisée pour séparer les données ;
- Les crochets tiennent les tableaux (brackets) ;
- Les accolades retiennent les objets.

Les types JSON : Number, String (entre guillemets), Boolean, Null, Array, Object

Exemple :

```
{    "nom"      :      "saidi",
    "prenom"  :      "ali",
    "age"     :      40,
    "interets" : null,
    "experience" : ["CSS", "JS", "HTML"],
    "adresse" : {"Rue" : "Sidi Maarouf", "Ville": "Casablanca", "codeP" : 10000 }
}
```

Fichier contact.json

Manipulation des données JSON

Pour traiter et afficher les données JSON dans les pages web, on a souvent besoin de les convertir en objets Javascript et vice versa.

- **Analyse syntaxique (parse)** : Convertir une chaîne de caractères en un objet natif.
- **Linéarisation (stringification)** : Convertir un objet natif en chaîne de caractères.

En Javascript, les méthodes utilisées sont :

- **JSON.parse** permet de convertir JSON vers un objet javascript.
- **JSON.stringify** permet de convertir des objets javascript vers des données JSON.

Exemples :

```
//Création d'un string JSON
var jsonData = '{"nom":"Saidi", "prenom":"Ali"}';
document.write(typeof(jsonData)+'<br>'); //string

//Convertir JSON vers Javascript
var jsObject = JSON.parse(jsonData);
document.write(typeof(jsObject)+'<br>');//object
document.write(jsObject+'<br>'); //[object object]

document.write(jsObject.nom + "
"+jsObject.prenom+'<br>'); //Saidi Ali
```

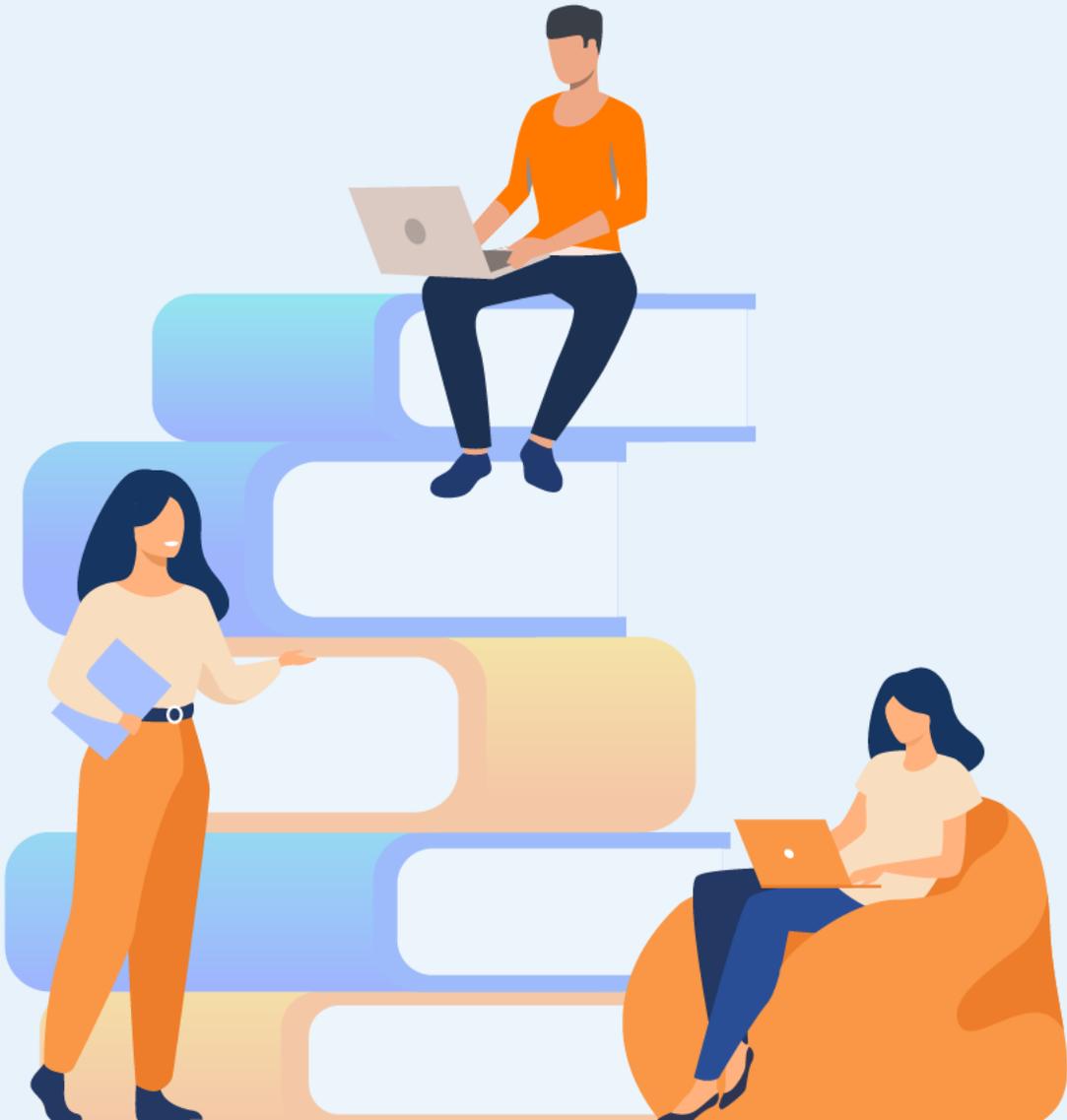
```
//Création d'un objet Javascript
var jsObject = {nom:"Saidi", prenom:"ali"};
document.write(typeof(jsObject)+'<br>');//Object

//Convertir javascript vers JSON
var jsonString = JSON.stringify(jsObject);
document.write(typeof(jsonString)+'<br>');//string

document.write(jsonString);//{"nom":"Saidi","prenom
":"ali"}
//remarquer la présence des "" dans les clés
```



WEBFORCE
BE THE CHANGE



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



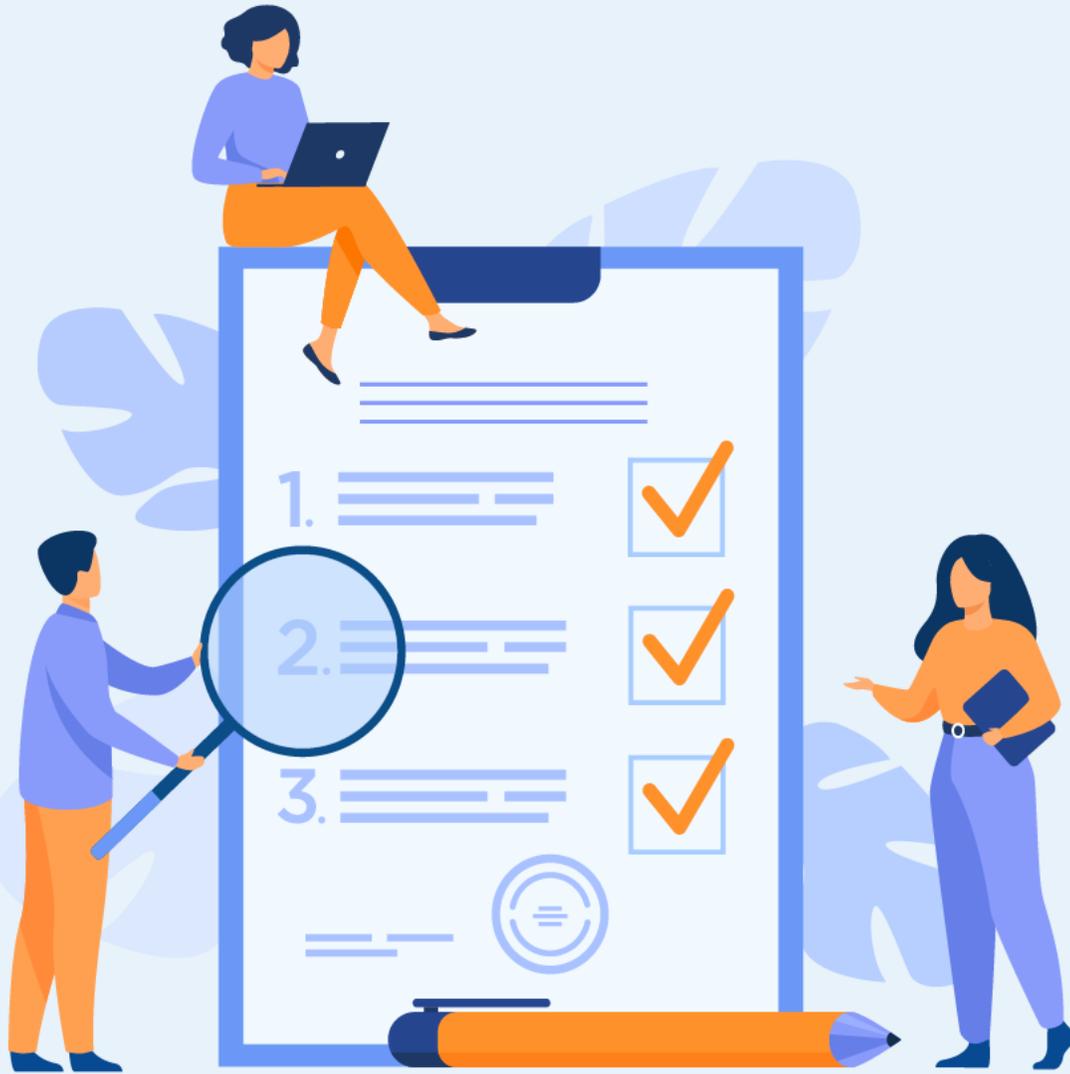
30 heures

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



WEBFORCE
BE THE CHANGE

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



Notion de l'arbre DOM

- **DOM** (Document Object Model) est une interface de programmation (API) normalisée par le W3C.
- Son rôle est d'accéder au contenu du navigateur web, et le modifier, en utilisant des scripts.
- Le DOM représente un document HTML sous forme d'un **arbre d'objets** (un paragraphe, une image, un style, etc).
- La modification du document HTML à l'aide du DOM consiste alors à ajouter ou supprimer des nœuds de l'arbre.
- DOM offre un ensemble de méthodes pour accéder aux éléments HTML.

Avec le modèle objet, JavaScript peut créer du contenu HTML dynamique en :

- Modifiant / Ajoutant / Supprimant les éléments HTML de la page ;
- Modifiant / Ajoutant / Supprimant les attributs des éléments HTML existants ;
- Modifiant les styles CSS de la page ;
- Réagissant aux événements HTML dans la page.

Notion de l'arbre DOM

Exemple : Soit le code HTML suivant correspondant à une page web. L'arbre DOM correspondant est représenté dans la figure ci-dessous :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Index</title>
</head>
<body>
  <h1>Ma page web</h1>
  <p>Bonjour, nous sommes les stagiaires de la filière développement
digital</p>
  <p>Nous étudions à l'<a href="http://www.ofppt.ma">OFPPT</a></p>
</body>
</html>
```

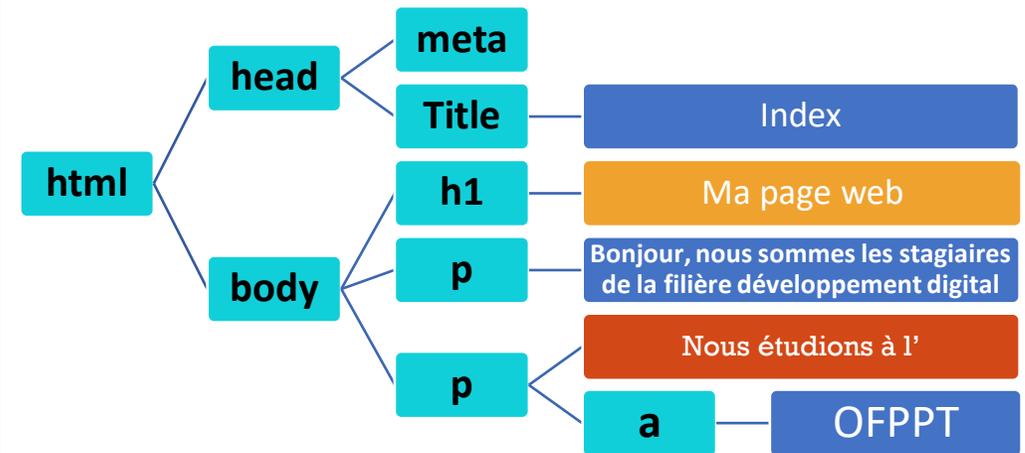


Figure 11: Arbre DOM d'une page HTML

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



Objet « document »

- L'objet **document** correspond à l'élément <html> de la page Web.
- La variable `document` est la racine du DOM.
- Cette variable est un objet et dispose des propriétés **head** et **body** qui permettent d'accéder respectivement aux éléments <head> et <body> de la page.

```
var h = document.head; // La variable h contient l'objet head du DOM
console.log(h);
var b = document.body; // La variable b contient l'objet body du DOM
console.log(b);
```

- L'objet **document** dispose d'un ensemble de méthodes et de propriétés permettant d'accéder et de manipuler le code html.

01 - Comprendre l'arbre dom, les nœuds parents et enfants

Objet Document



Méthodes de recherche d'éléments html

Méthode	Description
<code>document.getElementById(id)</code>	Retourne un élément par la valeur de l'attribut ID
<code>document.getElementsByTagName(name)</code>	Retourne les éléments par nom de balise
<code>document.getElementsByClassName(name)</code>	Retourne les éléments par nom de classe CSS

Exemple :

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <h1 id="p1" class="c1">cours DOM JS</h1>
  <p class="c1">1er paragraphe</p>
  <p class="c2">2ème paragraphe</p>
</body>
</html>
```

```
let e1=document.getElementById("p1");
console.log(e1);//<h1 id="p1" class="c1">cours DOM JS</h1>

let e2=document.getElementsByTagName("p");
console.log(e2);//[HTMLCollection(2) [p.c1, p]]

let e3=document.getElementsByClassName("c1");
console.log(e3);
//HTMLCollection(2) [h1#p1.c1, p.c1, p1: h1#p1.c1]
```

01 - Comprendre l'arbre dom, les nœuds parents et enfants

Objet Document



Méthodes d'ajout et suppression d'éléments

Méthode	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>	Écrire dans un document HTML
<code>document.getElementById(id).onclick = fonction(){code}</code>	Ajouter un événement de clic à l'élément sélectionné

01 - Comprendre l'arbre dom, les nœuds parents et enfants

Objet Document



Propriétés des éléments DOM

Méthode	Description
<code>element.innerHTML</code>	Permet de récupérer tout le contenu HTML d'un élément du DOM
<code>element.attribute</code>	Changer l'attribut d'un élément
<code>element.style.property</code>	Changer le style d'un élément
<code>element.textContent</code>	Renvoie tout le contenu textuel d'un élément du DOM, sans le balisage HTML
<code>element.classList</code>	Permet de récupérer la liste des classes d'un élément du DOM

CHAPITRE 1

Comprendre l'arbre dom, les nœuds parents et enfants

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



Relations entre les nœuds

Les éléments du DOM sont appelés des nœuds, qui sont en relation hiérarchique sous forme d'un arbre. Le nœud supérieur est appelé racine (ou nœud racine).

La relation entre les nœuds peut être qualifiée de relation :

- **Parent / Child:** des nœuds peuvent avoir des **ascendants** et des **descendants**
 - Nœuds ascendants sont les nœuds qui sont parents d'un nœud (ou parents d'un nœud parent) ;
 - Nœuds descendants sont les nœuds qui sont enfants d'un nœud (ou enfants d'un nœud enfant) ;
 - Chaque nœud a exactement un parent, sauf la racine.
 - Un nœud peut avoir plusieurs enfants.
- **Sibling :** correspondent aux frères d'un nœud, c'à-d les nœuds avec le même parent.

```
<ul>
  <li>1er élément</li>
  <li>2ème élément
    <p>paragraphe</p>
    <a>Lien</a>
  </li>
  <li>3ème élément</li>
</ul>
```

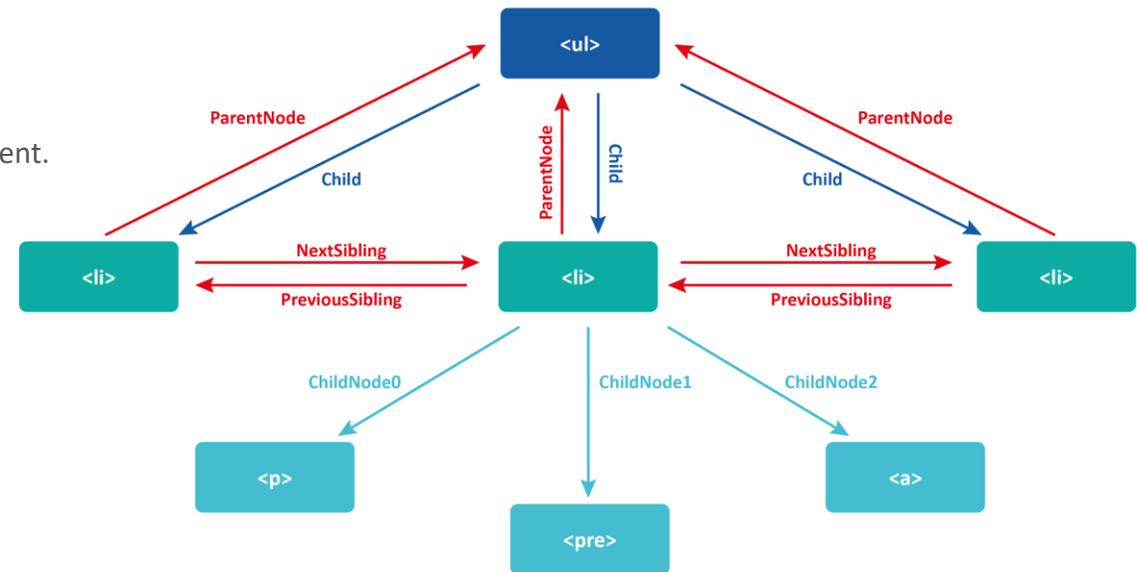


Figure 12 : Relation entre les nœuds

01 - Comprendre l'arbre dom, les nœuds parents et enfants

Navigation dans le DOM



Types de nœuds du DOM

Chaque objet du DOM a une propriété **nodeType** qui indique son type. La valeur de cette propriété est :

- `document.ELEMENT_NODE` pour un nœud "élément" (balise HTML).
- `document.TEXT_NODE` pour un nœud textuel.

Exemple :

```
if (document.body.nodeType === document.ELEMENT_NODE)
{
    console.log("Body est un nœud élément");
}
else
{
    console.log("Body est un nœud textuel");
}
//Body est un nœud élément
```

Types de nœuds du DOM

Chaque objet du **DOM** de type **ELEMENT_NODE** possède une propriété **childNodes** qui correspond à une collection de ses différents enfants :

- On peut connaître la taille de la collection avec la propriété **length** ;
- On peut accéder aux éléments grâce à leur indice ;
- On peut parcourir la collection avec une boucle **for**.



Remarque

- les retours à la ligne et les espaces entre les balises dans le code HTML sont considérés par le navigateur comme des nœuds textuels

Exemple :

```
console.log(document.body.childNodes[1]);  
//Affiche <h1 id="p1" class="c1">cours DOM JS</h1>  
  
// Afficher les noeuds enfant du noeud body  
for (var i = 0; i < document.body.childNodes.length; i++)  
    console.log(document.body.childNodes[i]);  
  
for(let i=0; i < document.body.childNodes[1].childNodes.length; i++)  
    console.log(` ${i} contient  
${document.body.childNodes[1].childNodes[i]} ` );  
// 0 contient [object Text]
```

```
▶ #text  
  <h1 id="p1" class="c1">cours DOM JS</h1>  
▶ #text  
  <p class="c1">1er paragraphe</p>  
▶ #text  
  <p class="c2">2ème paragraphe</p>  
▶ #text
```

Figure 13 : Résultat du code

01 - Comprendre l'arbre dom, les nœuds parents et enfants

Navigation dans le DOM



Types de nœuds du DOM

Chaque objet du **DOM** possède une propriété **parentNode** qui renvoie son nœud parent sous la forme d'un objet DOM.

Le parent de l'élément document est **null**.

Exemple :

```
console.log(document.parentNode); // Affiche null

var h1 = document.body.childNodes[1];

console.log(h1.parentNode); // Affiche le noeud body
```

Navigation entre les nœuds de l'arborescence DOM

- **Navigation dans les nœuds enfants**
 - **firstChild** : Retourne le premier enfant de l'élément.
 - **firstElementChild** : Retourne le premier élément enfant du parent.
 - **lastChild** : Retourne le dernier enfant de l'élément.
 - **lastElementChild** : Retourne le dernier élément enfant du parent.
 - **childNodes** : Retourne tous les enfants de l'élément sous forme d'une collection.
 - **children** : Renvoie tous les enfants qui sont des éléments sous forme d'une collection.
- **Navigation dans les nœuds parents**
 - **parentNode** : Renvoie le nœud parent de l'élément.
 - **parentElement** : Renvoie le nœud de l'élément parent de l'élément.
- **Navigation dans les nœuds frères**
 - **nextSibling** : Renvoie le nœud frère correspondant au prochain enfant du parent.
 - **nextElementSibling** : Renvoie l'élément frère correspondant au prochain enfant de son parent.
 - **previousSibling** : Renvoie le nœud frère qui est un enfant précédent de son parent.
 - **previousElementSibling** : Renvoie l'élément frère qui est un enfant précédent de son parent.

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `firstChild`

La propriété `firstChild`, appelée sur un nœud, retourne le premier nœud de l'élément. Ce nœud n'est pas nécessairement un élément, il peut également contenir du texte ou un commentaire.

Exemple :

```
<div id="parent">
  <h1>Un titre</h1>
  <p>Un paragraphe</p>
</div>

<script>
  let elt      = document.getElementById("parent");
  let premElt  = elt.firstChild;
  console.log(premElt); // text node
  console.log(premElt.nodeName); // #text
</script>
```



Remarque

- Re-exécuter ce code en supprimant l'espace entre l'élément « div » et la balise « h1 »

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `firstElementChild`

La propriété `firstElementChild` retourne le premier enfant, de type élément, du parent.

Exemple :

```
<div id="parent">
  <h1>Un titre</h1>
  <p>Un paragraphe</p>
</div>

<script>
  let element = document.getElementById("parent");
  let premElt = element.firstElementChild.nodeName;
  console.log(premElt); // h1
</script>
```

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- lastChild

La propriété **lastChild** permet de sélectionner le dernier enfant de l'élément parent. Elle renvoie **null** s'il n'y a pas d'enfant.

Exemple :

```
<div id="parent">
  <h1>Un titre</h1>
  <p>Un paragraphe</p>
</div>

<script>
  let element = document.getElementById("parent");
  let DernElt = element.lastChild.nodeName;
  console.log(DernElt); // #text
</script>
```

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `lastElementChild`

La propriété `lastElementChild` retourne le dernier enfant, de type élément, du parent.

Exemple :

```
<div id="parent">
  <h1>Un titre</h1>
  <p>Un paragraphe</p>
</div>

<script>
  let element = document.getElementById("parent");
  let DernElt = element.lastElementChild.nodeName;
  console.log(DernElt); // P
</script>
```

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `childNodes`

La propriété `childNodes` d'un nœud retourne une liste de nœuds enfants d'un élément donné. Les indices des éléments enfants commencent à 0.

Exemple :

```
<div id="parent">
  <h1>Un titre</h1>
  <p>Un paragraphe</p>
  <a href="#">Un lien</a>
</div>

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

La liste « `childNodes` » comprend les nœuds `'#text'` et `'element'`.

▶ `#text`

`<h1>Un titre</h1>`

▶ `#text`

`<p>Un paragraphe</p>`

▶ `#text`

`link`

▶ `#text`

Figure 14 : Résultat du code

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- children

La propriété children, appelée sur l'élément parent, permet d'obtenir uniquement les nœuds de type élément.

Exemple :

```
<div id="parent">
  <h1>Un titre</h1>
  <p>Un paragraphe</p>
  <a href="#">Un lien</a>
</div>

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

H1 - Un titre

P - Un paragraphe

A - link

Figure 15 : Résultat du code

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- parentNode

La propriété **parentNode** retourne l'élément parent de l'élément appelant ou **null** (si le parent n'existe pas).

Exemple :

```
<div id="parent">
  <h1 id="id1">Un titre</h1>
  <p>Un paragraphe</p>
  <a href="#">Un lien</a>
</div>

<script>
  let element = document.getElementById("id1");
  let parent = element.parentNode;
  console.log("élément parent : " +
parent.nodeName);
  console.log(parent);
</script>
```

élément parent : BODY

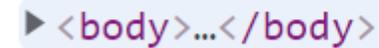


Figure 16 : Résultat du code

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `parentElement`

La propriété `parentElement` renvoie le parent de l'élément. La différence entre `parentNode` et `parentElement` est montrée dans l'exemple suivant :

```
<div id="parent">
  <h1 id="id1">Un titre</h1>
  <p>Un paragraphe</p>
  <a href="#">Un lien</a>
</div>

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

Élément parent - BODY

▶ #document

null

Figure 17 : Résultat du code

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `nextSibling`

La propriété `nextSibling` permet d'accéder à l'élément frère d'un élément. L'élément retourné n'est pas nécessairement un nœud d'élément.

Exemple :

```
<div id="parent">
  <h1 id="id1">Un titre</h1>
  <p>Un paragraphe</p>
  <a href="#">Un lien</a>
</div>

<script>
  let element = document.getElementById("id1");
  let next = element.nextSibling;
  console.log("Élément frère suivant - " + next.nodeName);
  console.log(next); </script>
```

Élément frère suivant - #text

▶ #text

Figure 18: Résultat du code

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `nextElementSibling`

La propriété `nextElementSibling` permet d'obtenir le nœud d'élément immédiatement suivant de l'élément appelant :

Exemple :

```
<div id="parent">
  <h1 id="id1">Un titre</h1>
  <p>Un paragraphe</p>
  <a href="#">Un lien</a>
</div>

<script>
  let element = document.getElementById("id1");
  let EltSuiv = element.nextElementSibling;
  console.log("Élément frère suivant - " + EltSuiv.nodeName);
  console.log(EltSuiv);
</script>
```

Élément frère suivant - P

`<p>Un paragraphe</p>`

Figure 19 : Résultat du code

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `previousSibling`

La propriété `previousSibling` appelée sur un élément, permet d'obtenir le nœud précédent.

Exemple :

```
<div id="parent">
  <h1 id="id1">Un titre</h1>
  <p>Un paragraphe</p>
  <a href="#">Un lien</a>
</div>

<script>
  let element = document.getElementById("id1");
  let eltPrec = element.previousSibling;
  console.log("Element frère précédent - " + eltPrec.nodeName);
  console.log(eltPrec);
</script>
```

Element frère précédent - #text

▶ #text

Figure 20 : Résultat du code

Navigation entre les nœuds de l'arborescence DOM (Exemples)

- `previousElementSibling`

La propriété `previousElementSibling` appelée sur un élément, permet d'obtenir le nœud précédent (de type élément).

Exemple :

```
<div id="parent">
  <h1 id="id1">Un titre</h1>
  <p id="id2">Un paragraphe</p>
  <a href="#">Un lien</a>
</div>
<script>
  let element = document.getElementById("id2");
  let eltPrec = element.previousElementSibling;
  console.log("Element frère précédant - " + eltPrec.nodeName);
  console.log(eltPrec);
</script>
```

Element frère précédant - H1

```
<h1 id="id1">Un titre</h1>
```

Figure 21 : Résultat du code



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

1. **Sélecteurs (simples, multiples...)**
2. Modes d'Accès aux éléments



02 - Connaître les bases de la manipulation du dom en javascript

Sélecteurs (simples, multiples...)



Sélecteurs CSS

En javascript, on peut chercher les éléments par leur sélecteur CSS :

- La méthode **querySelector()** renvoie le premier élément qui correspond à un ou plusieurs sélecteurs CSS spécifiés.
- La méthode **querySelectorAll()** renvoie tous les éléments correspondants.

Syntaxe :

```
document.querySelector(sélecteur CSS)  
document.querySelectorAll(sélecteur CSS)
```

02 - Connaître les bases de la manipulation du dom en javascript

Sélecteurs (simples, multiples...)



La méthode `querySelector()`

Exemples :

- //Obtenir le premier élément `<p>` dans le document :

```
document.querySelector("p");
```
- //Obtenir le premier élément `<p>` du document qui a `class="par"`:

```
document.querySelector("p.par");
```
- //Modifier le texte de l'élément dont l'attribut `id="id1"`:

```
document.querySelector("#id1").innerHTML = "Bonjour!";
```
- //Obtenir le premier élément `<p>` dans le document où le parent est un élément `<div>` :

```
document.querySelector("div > p");
```
- //Obtenir le premier élément `<a>` dans le document qui a un attribut `"target"`:

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

02 - Connaître les bases de la manipulation du dom en javascript

Sélecteurs (simples, multiples...)



La méthode `querySelectorAll()`

Exemples :

- //Obtenir tous les éléments `<p>` du document et définir la couleur d'arrière-plan du premier élément `<p>` (index 0) :

```
let x = document.querySelector("p");  
x[0].style.backgroundColor = "red";
```

- //Obtenir tous les éléments `<p>` du document qui ont l'attribut `class="par"`, et définir l'arrière-plan du premier élément `<p>` :

```
let x = document.querySelector("p.par");  
x[0].style.backgroundColor = "red";
```

- // Calculer le nombre d'éléments qui ont l'attribut `class="par"` (en utilisant la propriété `length` de l'objet `NodeList`) :

```
var x = document.querySelectorAll(".par").length;  
// Définir la couleur d'arrière-plan de tous les éléments du document qui ont l'attribut class="par":
```

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

- //Définir la bordure de tous les éléments `<a>` du document qui ont un attribut `"target"`:

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

02 - Connaître les bases de la manipulation du dom en javascript

Sélecteurs (simples, multiples...)



La méthode `querySelectorAll()`

Exemples:

- //Sélectionner le premier paragraphe du document et modifier son texte avec la propriété `textContent` */
`document.querySelector('p').textContent = '1er paragraphe du document';`
`let documentDiv = document.querySelector('div'); //1er div du document`
- //Sélectionner le premier paragraphe du premier div du document et modifier son texte
`documentDiv.querySelector('p').textContent = '1er paragraphe du premier div';`
- //Sélectionner le premier paragraphe du document avec un attribut `class='bleu'` et changer sa couleur en bleu avec la propriété `style` */
`document.querySelector('p.bleu').style.color = 'blue';`
- //Sélectionne tous les paragraphes du premier div
`let divParas = documentDiv.querySelectorAll('p');`

CHAPITRE 2

Connaître les bases de la manipulation du dom en javascript

1. Sélecteurs (simples, multiples...)
2. **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 du DOM

On peut Chercher les éléments directement par nom en utilisant les méthodes suivantes :

- `document.getElementsByTagName()`
- `document.getElementsByClassName ()`
- `document.getElementById ()`
- `document.getElementsByName ()`

Ou bien en utilisant un sélecteur CSS associé :

- `document.querySelector()`
- `document.querySelectorAll()`

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()` renvoie un objet qui représente l'élément dont la valeur de l'attribut `id` correspond à la valeur spécifiée en argument.

```
//Sélectionner 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

- La méthode `getElementsByClassName()` renvoie une liste des éléments possédant un attribut `class` avec la valeur spécifiée en argument.

```
//Sélectionner les éléments avec une class = 'bleu'  
let bleu = document.getElementsByClassName('bleu');  
for(valeur of bleu){          valeur.style.color = 'blue'; }
```

Accéder à un élément en fonction de son identité (Nom de la balise)

- La méthode `getElementsByTagName()` permet de sélectionner des éléments en fonction de leur nom.

```
//Sélectionner tous les éléments p du document  
let paras = document.getElementsByTagName('p');  
for(valeur of paras){          valeur.style.color = 'blue'; }
```

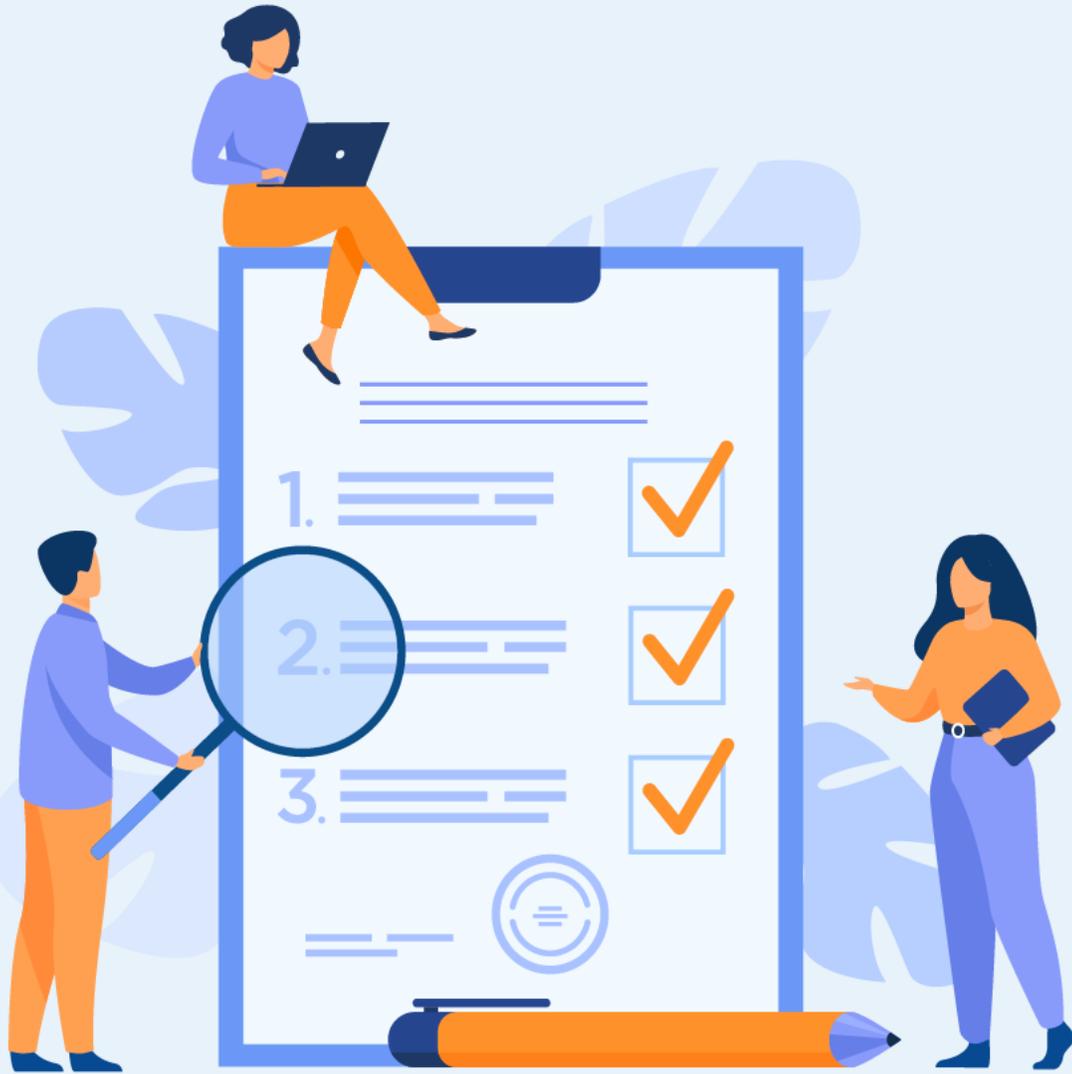
Accéder directement à des éléments particuliers avec les propriétés de Document

L'API DOM fournit également des propriétés permettant d'accéder directement à certains éléments du document. Parmi ces propriétés on trouve :

- 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 attribut href avec une valeur ;
- La propriété **title** qui retourne le titre (le contenu de l'élément title) du document ;
- La propriété **url** qui renvoie l'URL du document sous forme d'une 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 d'une chaîne de caractères.

Exemple :

```
//Sélectionner l'élément body et appliquer une couleur bleu  
document.body.style.color = 'blue';  
  
//Modifier le texte de l'élément title  
document.title= 'Le DOM';
```



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



Créer un élément en JavaScript

La méthode `createElement` permet de créer de nouveaux éléments dans le document.

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



Remarque

- L'élément créé par la méthode `createElement()` ne s'attache pas automatiquement au document

Exemples :

```
let element1 = document.createElement('p');  
console.log(element1); // <p></p>
```

```
let element2 = document.createElement('div');  
console.log(element2); // <div></div>
```

La méthode `createElement` convertit le nom de l'élément en minuscule

```
let element3 = document.createElement('DIV');  
console.log(element3); // <div></div>
```

03 - Manipuler les éléments html

Manipulation des éléments



Ajouter un élément en JavaScript

Pour ajouter un élément à l'arborescence du DOM (après l'avoir créé), il faut l'attacher à un élément parent.

La méthode **append()** insère un objet en tant que dernier enfant d'un élément parent.

Exemple :

```
let parent = document.getElementById("parent"); // sélectionner un élément parent
let enfant = document.createElement("p"); // Créer un élément enfant
enfant.innerHTML = "C'est un nouveau élément"; // Ajouter un texte à l'élément créé
parent.append(enfant); // Attacher l'enfant à l'élément parent
```

Supprimer un élément en JavaScript

La méthode `removeChild()` supprime un élément de la structure du DOM. Le nœud à supprimer est passé en argument à la méthode. Une référence vers le nœud supprimé est retournée à la fin.

Exemple :

```
let parent = document.getElementById("parent"); // sélectionner un élément parent
let enfant = document.getElementById("eltSupp"); // Sélectionner un élément enfant
parent.removeChild(enfant);
```

Modifier un élément en JavaScript

La méthode `replaceChild()` remplace un nœud par un autre nœud dans le DOM. Une référence vers le nœud remplacé est retournée à la fin.

Syntaxe :

```
parent.replaceChild(nouveauElement, ancienElement)
```

Exemple :

```
let parent = document.getElementById("parent"); // sélectionner un élément parent
let AncienElement = document.getElementById("id1"); // sélectionner l'ancien élément
let nouvelElement = document.createElement("h2"); // Créer un nouveau élément de type <h2>
nouvelElement.innerHTML = "C'est le nouveau élément."
parent.replaceChild(nouvelElement, AncienElement);
```

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

Les propriétés `.style` ou `.className` appliquées sur un élément permettent de changer les styles CSS.

Exemple :

```
<div>
  <div>
    <label>Nom : </label><br>
    <input type="text" class="style1" id="b1">
  </div>
</div>

<script>
// Modifier le style de l'élément qui a la l'attribut class=style1
document.getElementsByClassName("style1").style.borderColor = "red";
</script>
```

03 - Manipuler les éléments html

Mise à jour des styles, attributs et classes



Définir le style à l'aide de `element.className`

La propriété `element.className` permet de changer les paramètres de style d'un élément HTML en lui attribuant une nouvelle classe dont le nom est passé à l'élément sélectionné.

```
<div>
  <div>
    <label>Nom : </label><br>
    <input type="text" class="style1" id="b1">
  </div>
</div>

<script>
// Modifier le style de l'élément qui a l'attribut class=style1 en lui associant une classe nommée
styleErreur
  document.getElementsByClassName("style1").className = "styleErreur";
</script>
```

Mise à jour d'un attribut avec `setAttribute`

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

Si l'attribut existe déjà, sa valeur est mise à jour. Sinon, un nouvel attribut est ajouté avec le nom et la valeur spécifiés.

Exemple 1 : Ajouter les attributs `class` et `disabled` à l'élément `<button>`

```
<button type="button" id="Btn">Click</button>

<script>
  // sélectionner l'élément
  let btn = document.getElementById("Btn");

  // Ajouter les attributs
  btn.setAttribute("class", "style1");
  btn.setAttribute("disabled", "");
</script>
```

Exemple 2 : Mettre à jour la valeur de l'attribut `href` de l'élément `<a>`.

```
<a href="#" id="lien">OFPPT</a>

<script>
  // sélectionner l'élément
  let lien = document.getElementById("lien");

  // Modifier la valeur de l'attribut href
  lien.setAttribute("href", "https://www.ofppt.ma");
</script>
```

03 - Manipuler les éléments html

Mise à jour des styles, attributs et classes



Suppression d'attributs d'éléments

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

Exemple : Supprimer l'attribut href d'un lien.

```
<a href="https://www.ofppt.com/" id="lien">OFPPT</a>

<script>
  // sélectionner l'élément
  let lien = document.getElementById("lien");

  // Supprimer la valeur de l'attribut href
  lien.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 DOMMenu Object



Création d'un DOMMenu Objet

L'objet **DOMMenu** en HTML représente l'élément <menu>.

Syntaxe :

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

Les attributs :

- **Label** : prend une valeur textuelle, spécifie le label du menu.
- **Type** : prend l'une des valeurs (list, toolbar, contex). Son rôle est de spécifier le type du menu à afficher.



Remarque

- Cet élément n'est plus supporté par les principaux navigateurs

03 - Manipuler les éléments html

Création DOMMenu Object



Création d'un DOMMenu Objetc

Exemple :

```
<menu type="context" id="monMenu">
  <menuitem label="Actualiser" onclick="window.location.reload();" icon="ico_reload.png"></menuitem>
  <menu label="Partager sur...">
    <menuitem label="Twitter" icon="ico_twitter.png" onclick="window.open('//twitter.com/intent/tweet?text=
'+window.location.href);">
    </menuitem>
    <menuitem label="Facebook" icon="ico_facebook.png" onclick="window.open('//facebook.com/sharer/sharer.
php?u='+window.location.href);">
    </menuitem>
  </menu>
  <menuitem label="Envoyer cette page" onclick="window.location='mailto:?body='+window.location.href;">
  </menuitem>
</menu>
```



WEBFORCE
BE THE CHANGE



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



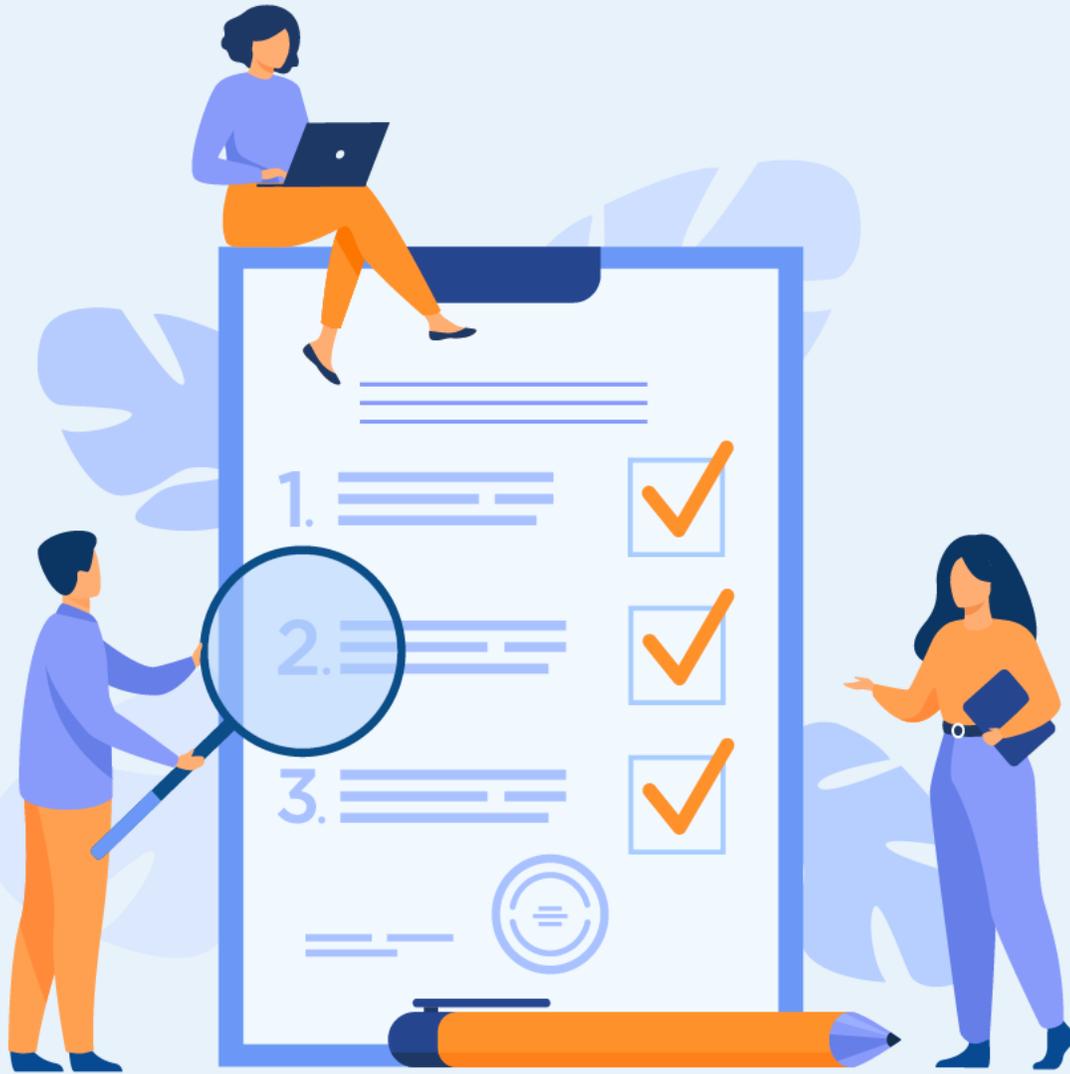
12 heures

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



07 heures

CHAPITRE 1

Comprendre la notion d'événement pour gérer l'interactivité

1. **Définition d'un événement**
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 ?

Un événement est une action effectuée soit par l'utilisateur soit par le navigateur.

Il existe plusieurs types d'événements : événement de souris, un événement de clavier, un événement de document, etc.

Exemples d'événements DOM :

- Clic sur un bouton par l'utilisateur ;
- Déplacement de la souris ;
- Chargement de la page ;
- Clic sur une touche du clavier ;
- Soumissions d'un formulaire ;
- ...

Javascript offre des mécanismes de réaction aux événements.

Les événements sont généralement traités par une fonction, qui s'exécute après que l'événement soit produit.

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 (Event Listener) :

L'écouteur d'événement est un objet qui attend qu'un certain événement se produise (un clic, un mouvement de souris, etc.).

2

Gestionnaire d'événements :

Le gestionnaire d'événements correspond généralement à une fonction appelée suite à la production de l'événement.

On distingue deux méthodes pour la gestion des événements en JavaScript :

- **Utilisation de l'attribut HTML :** Attacher directement un événement à un élément HTML en tant qu'attribut.

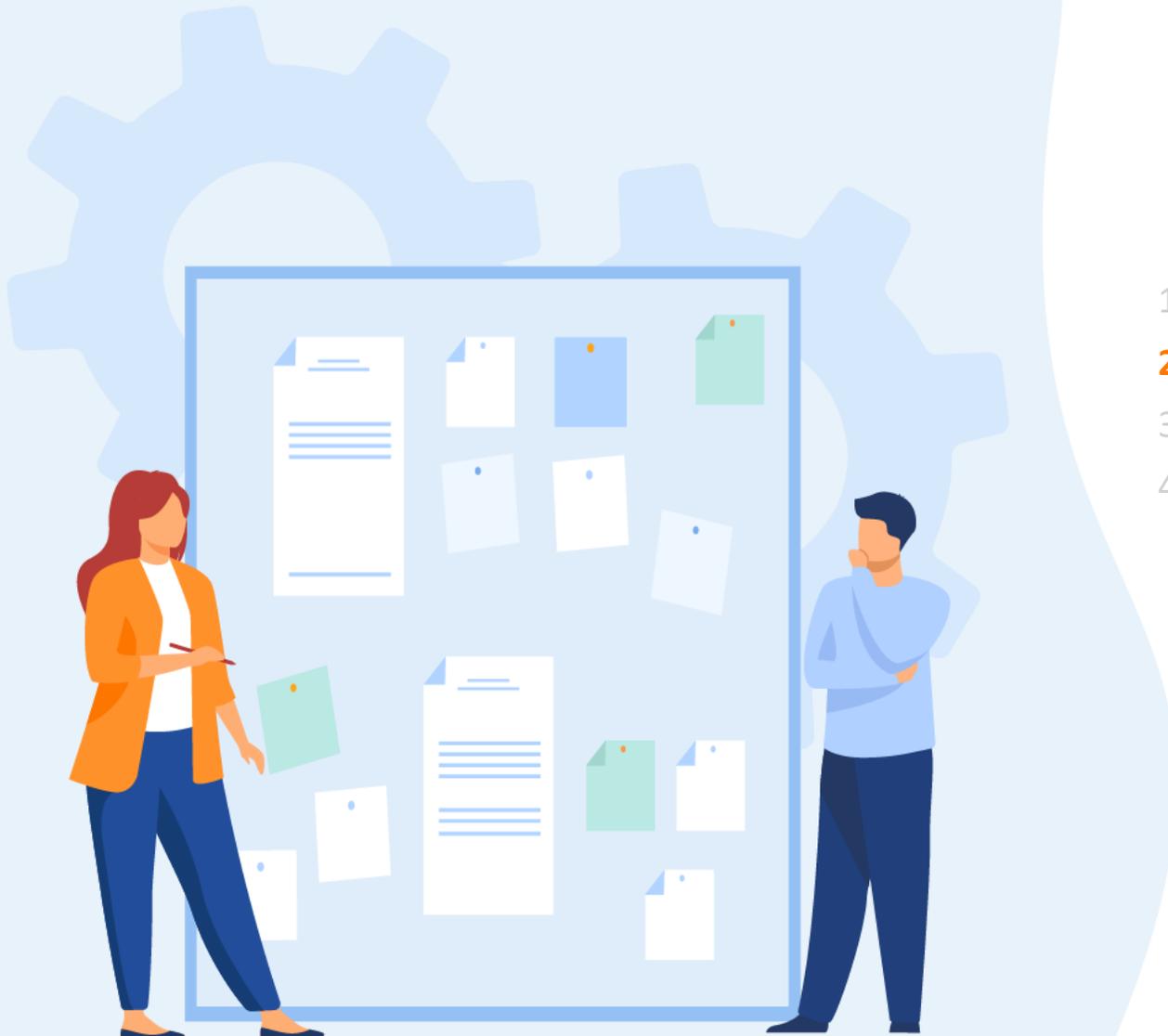
Exemple : `<button onclick="alert('Bonjour')">Clic</button>`

Utilisation de la méthode **addEventListener** : Associer l'événement à l'élément en utilisant 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` appliquée sur un élément DOM, lui ajoute un gestionnaire pour un événement particulier.

Cette fonction sera appelée à chaque fois que l'événement se produit sur l'élément.



Remarque

- On peut ajouter plusieurs événements à un même élément.

Syntaxe : la méthode prend deux paramètres : le type de l'événement et la fonction qui gère l'événement.

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

Élément : un élément HTML auquel l'événement est attaché.

Événement : le nom de l'événement.

Fonction de rappel : la fonction qui va gérer l'événement.

UseCapture : paramètre booléen facultatif qui prend par défaut la valeur `false` (spécifie s'il faut utiliser le bouillonnement d'événements ou la capture d'événements).

01 - Comprendre la notion d'événement pour gérer l'interactivité

Méthode `addEventListener`



Méthode `addEventListener()`

Exemple 1 :

```
let element = document.getElementById("btn");

element.addEventListener("click", message);

//Fonction qui gère l'événement
function message() {
  alert("Vous avez cliqué!")
}
```

Exemple 2 : utiliser une fonction interne dans la méthode `addEventListener()`

```
let element = document.getElementById("btn");
element.addEventListener("click", function () { alert("Vous avez cliqué!");});
```

01 - Comprendre la notion d'événement pour gérer l'interactivité

Méthode `addEventListener`



Événements multiples utilisant `addEventListener()`

La méthode `addEventListener()` permet d'ajouter plusieurs méthodes identiques ou différentes à un seul élément. Ainsi, il est possible d'ajouter plus de deux écouteurs d'événement pour le même événement.

Exemple :

```
let element = document.getElementById("btn");
element.addEventListener("click", fct1);
element.addEventListener("click", fct2);

function fct1() {
  alert("Fonction 1");
}

function fct2() {
  alert("Fonction 2");
}
```

01 - Comprendre la notion d'événement pour gérer l'interactivité

Méthode `addEventListener`



Événements multiples utilisant `addEventListener()`

La méthode `addEventListener()` permet aussi d'attacher plusieurs types d'événements **différents** au même élément HTML .

Exemple :

```
let element = document.getElementById("btn");
element.addEventListener("click", clickFct); //Événement : clic de la souris
element.addEventListener("mouseover", mouseoverFxn); //Événement : passage de la souris sur un élément
element.addEventListener("mouseout", mouseoutFxn); //Événement : la souris quitte l'élément

function clickFct() {
    alert("Vous avez cliqué :");
}
function mouseoverFxn() {
    element.style.background = "red";
    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

La méthode `removeEventListener()` permet de supprimer l'écouteur d'événement d'un élément ou un objet HTML.

Exemple :

```
<p class="style1">Cet élément possède l'événement "mouseover"</p>
<button id="btn" onclick="SupprimerEvt()">Supprimer l'événement</button>

<script>
  let element = document.querySelector(".style1"); //Sélectionner l'élément button
  element.addEventListener("mouseover", fct1); // Ajouter un événement de type « mouseover »

function fct1(){
  alert("Evénement déclenché!");
}

function SupprimerEvt(){
  element.removeEventListener("mouseover", fct1);
}
</script>
```

CHAPITRE 1

Comprendre la notion d'événement pour gérer l'interactivité

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



01 - Comprendre la notion d'événement pour gérer l'interactivité

MouseEvents

Les événements de la souris JavaScript

Les événements de la souris sont déclenchés quand elle interagit avec les éléments de la page.

Les événements DOM définissent neuf types d'événements de la souris : **mousedown**, **mouseup**, et **click**.

Quand la souris est enfoncée, trois événements se déclenchent dans l'ordre suivant :

- **Mousedown** : se déclenche lorsqu'on appuie sur le bouton gauche de la souris.
- **Mouseup** : se déclenche lorsqu'on relâche le bouton de la souris.
- **Click** : se déclenche lorsqu'un événement **mousedown** et un **mouseup** sont détectés sur l'élément.

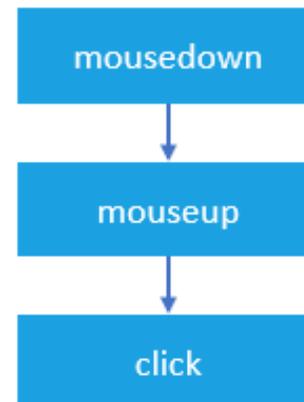


Figure 22 : Ordre des événements de la souris

01 - Comprendre la notion d'événement pour gérer l'interactivité

MouseEvents

Les événements de la souris JavaScript

- **dblclick**

L'événement **dblclick** se déclenche lorsqu'on double-clique sur un élément en utilisant la souris.

Un événement **dblclick** est déclenché par deux événements **click**.

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**

Les événements **click** ont toujours lieu avant l'événement **dblclick**.

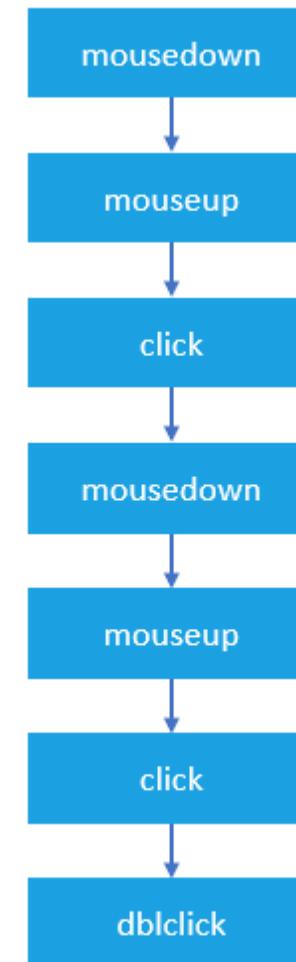


Figure 23 : Ordre de tous les événements de la souris

01 - Comprendre la notion d'événement pour gérer l'interactivité

MouseEvents



Les événements de la souris JavaScript

- **mousemove**

L'événement **mousemove** se déclenche à plusieurs reprises lorsqu'on déplace le curseur de la souris autour d'un élément.

Pour ne pas ralentir la page, le gestionnaire de l'événement **mousemove** n'est enregistré qu'en cas de besoin, et doit être supprimé dès qu'il n'est plus utilisé.

Exemple :

```
element.onmousemove = fct1;  
element.onmousemove = null;
```

- **mouseover / mouseout**

L'événement **mouseover** se déclenche lorsque le curseur de la souris se déplace à l'extérieur d'un élément.

L'événement **mouseout** se déclenche lorsque le curseur de la souris survole un élément, puis se déplace vers autre élément.

- **mouseenter / mouseleave**

L'événement **mouseenter** se déclenche lorsque le curseur de la souris se déplace de l'extérieur à l'intérieur d'un élément.

L'événement **mouseleave** se déclenche lorsque le curseur de la souris se déplace de l'intérieur à l'extérieur d'un élément.

01 - Comprendre la notion d'événement pour gérer l'interactivité

MouseEvents



Enregistrement des gestionnaires d'événements de souris

- Étape 1 : sélectionner l'élément concerné par l'événement à l'aide des méthodes de sélection (`querySelector()` ou `getElementById()`).
- Étape 2 : enregistrer l'événement de la souris à l'aide de la méthode `addEventListener()`.

Exemple : soit le bouton suivant :

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

Le code suivant associe l'événement de clic au bouton précédent :

```
let btn = document.querySelector('#btn');  
  
btn.addEventListener('click', (event) => {  
  console.log('click');  
});
```

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 .



Figure 24 : Boutons de la souris

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 (dans l'écran) correspondant à l'emplacement de la souris par rapport à l'ensemble de l'écran.
- Les propriétés `clientX` et `clientY` fournissent les coordonnées horizontales et verticales dans la zone cliente de l'application où l'événement de la souris s'est produit.

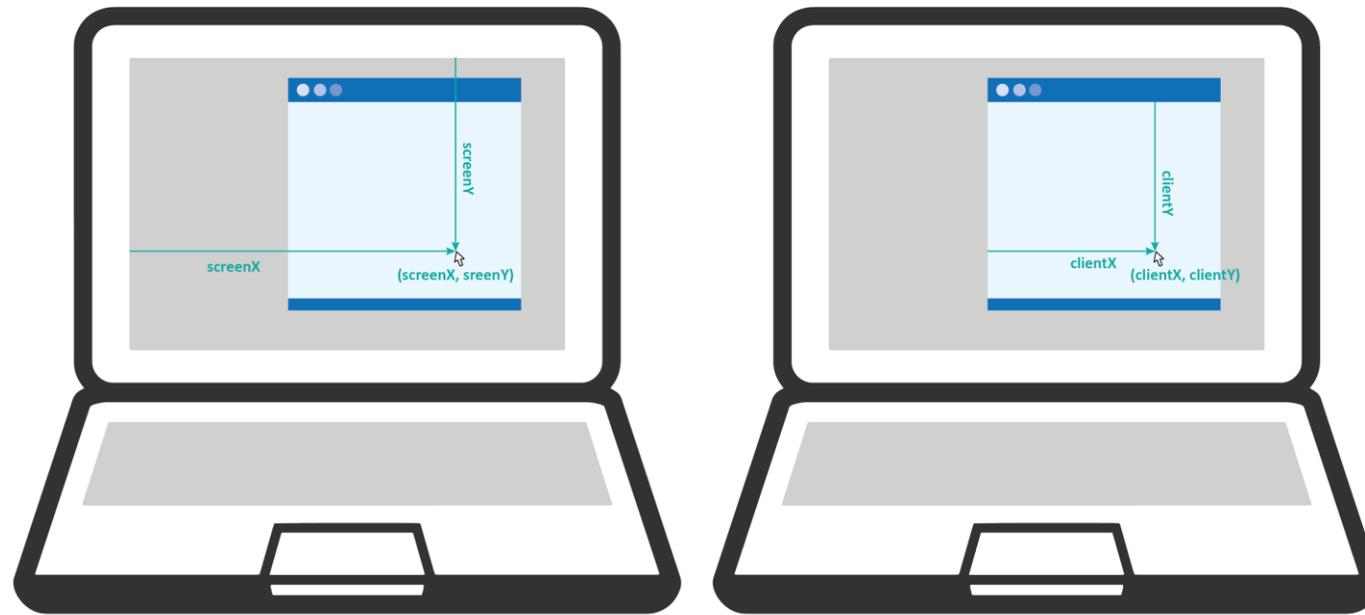


Figure 25 : Propriétés `screen(x,y)` et `client(x,y)`

01 - Comprendre la notion d'événement pour gérer l'interactivité

MouseEvents



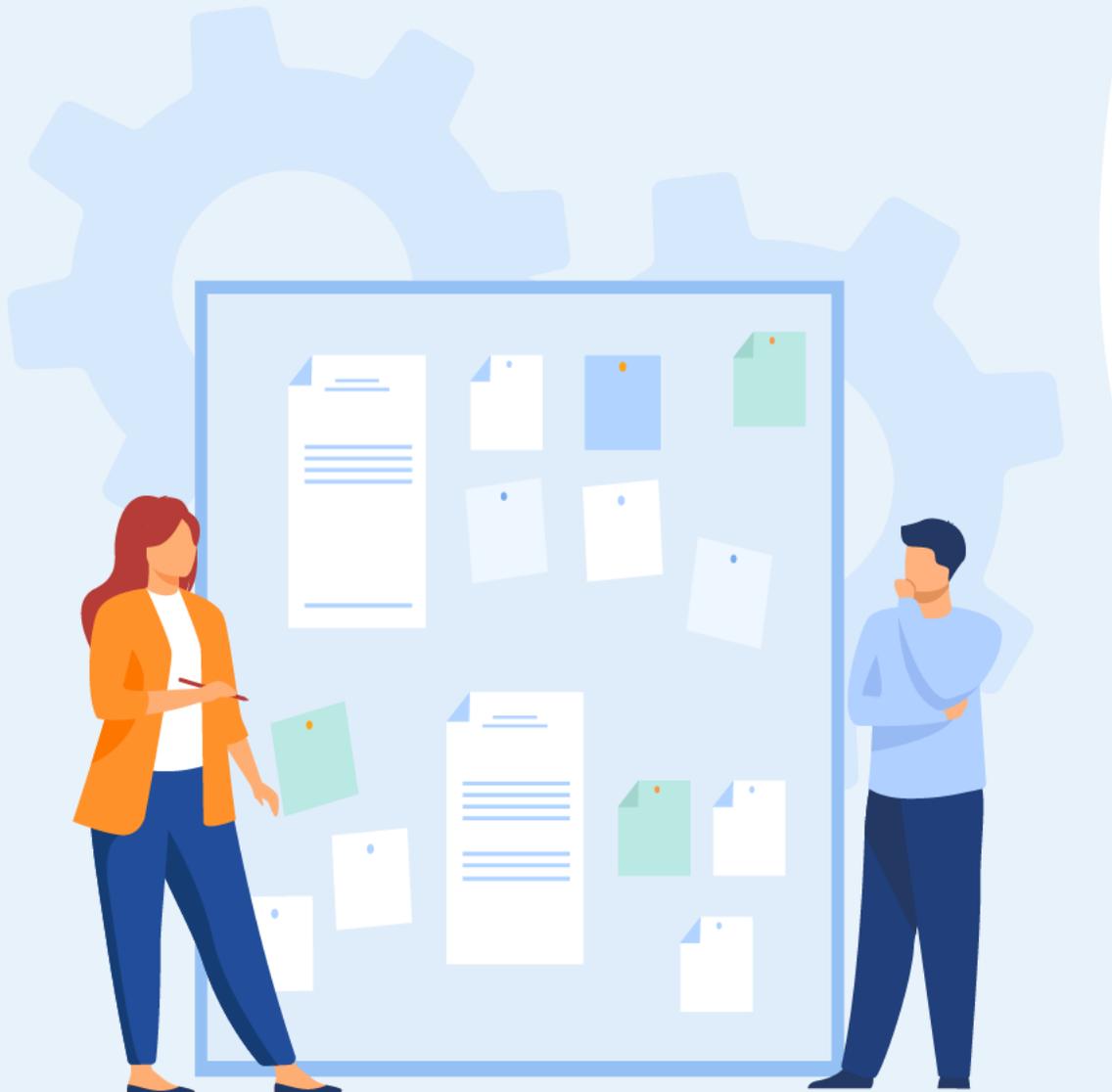
Exemple :

```
<!DOCTYPE html>
<html>
<head>
  <title>Les événements de la souris</title>
  <style>
    #pos { background-color: goldenrod; height: 200px; width: 400px; }
  </style>
</head>
<body>
  <p>Faire bouger la souris pour voir sa position.</p>
  <div id="pos"></div>
  <p id="affichage"></p>
  <script>
    let pos = document.querySelector('#pos');
    pos.addEventListener('mousemove', (e) => {
      let affichage = document.querySelector('#affichage');
      affichage.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



Les événements du clavier JavaScript

Il existe trois principaux événements du clavier :

- **Keydown** : déclenché lorsqu'on appuie sur une touche du clavier. Cet événement se déclenche plusieurs fois pendant que la touche est enfoncée.
- **Keyup** : déclenché lorsqu'on relâche une touche du clavier.
- **Keypress** : déclenché lorsqu'on appuie sur les caractères comme a, b, ou c, et pas sur les touches fléchées gauche, etc. Cet événement se déclenche également à plusieurs reprises lorsqu'on maintient la touche du clavier enfoncée.

Lorsqu'on appuie sur une touche du clavier, les trois événements sont déclenchés dans l'ordre suivant :

1. **keydown**
2. **keypress**
3. **Keyup**



Remarques

- Les événements **keydown** et **keypress** sont déclenchés avant toute modification apportée à la zone de texte.
- L'événement **keyup** se déclenche après que les modifications aient été apportées à la zone de texte.

01 - Comprendre la notion d'événement pour gérer l'interactivité

Interaction avec le clavier



Gestion des événements du clavier

Étape 1 : sélectionner l'élément sur lequel l'événement clavier se déclenchera (généralement d'une zone de texte).

Étape 2 : associer à l'élément la méthode `addEventListener()` pour enregistrer un gestionnaire d'événements.

Exemple : soit la zone de texte suivante :

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

Dans le code suivant, les trois gestionnaires d'événements seront appelés à l'enfoncement d'une touche de caractère :

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

01 - Comprendre la notion d'événement pour gérer l'interactivité

Interaction avec le clavier



Les propriétés de l'événement clavier

L'événement clavier possède deux propriétés importantes :

- **key** : renvoie le caractère qui a été enfoncé.
- **Code** : renvoie le code de touche physique.

Par exemple, le clic sur la touche z du clavier : **event.key** retourne « z » et les « **event.code** » retourne « **KeyZ** ».

Exemple :

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

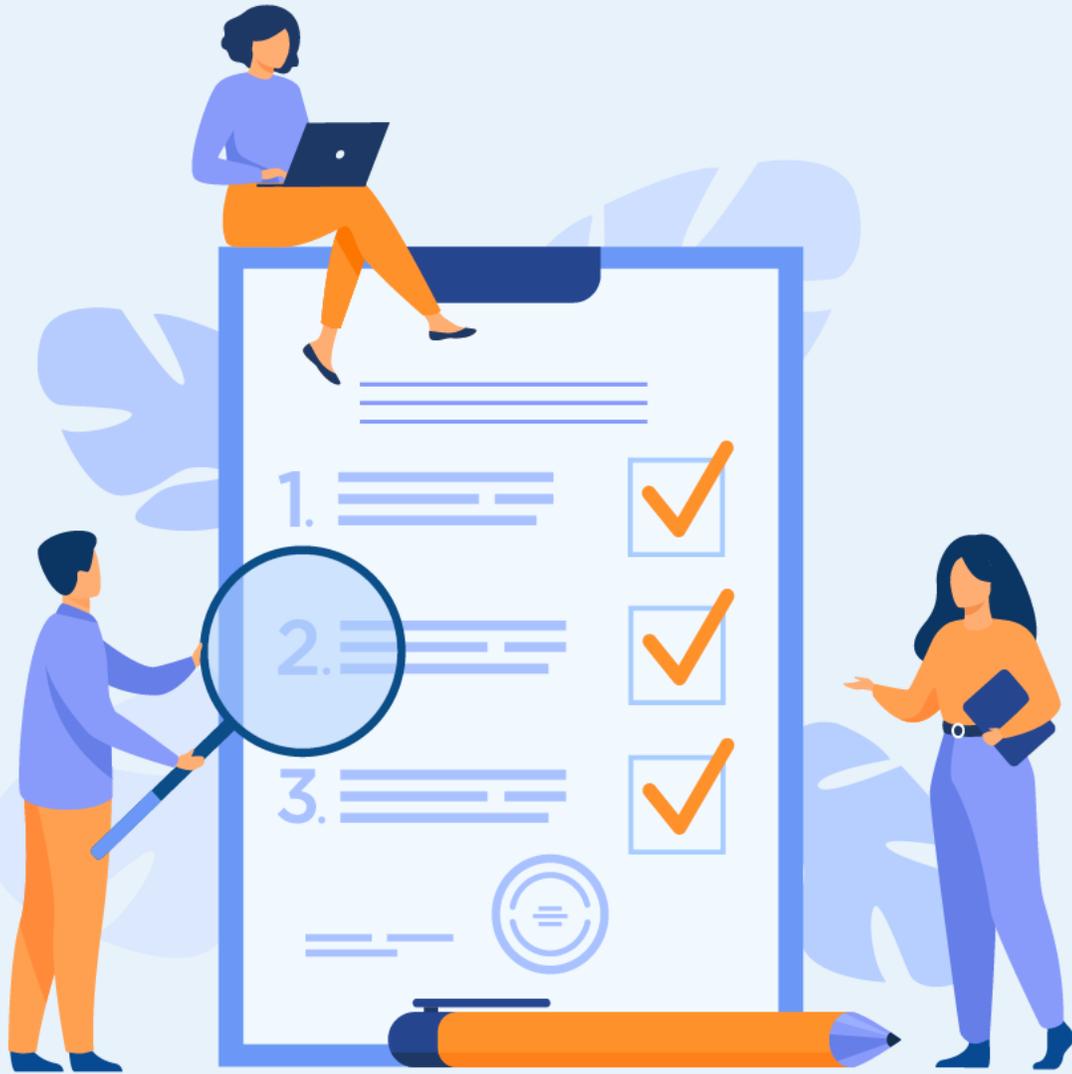
<script>
  let zone = document.getElementById('message');
  zone.addEventListener('keydown', (event) => {
    console.log(`key=${event.key},code=${event.code}`);
  });
</script>
```

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



05 heures

CHAPITRE 2

Gérer les éléments d'un formulaire

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



Soumission d'un événement

La méthode `submit()` du DOM est utilisée pour soumettre les données du formulaire à l'adresse spécifiée dans l'attribut `action`.

Cette méthode se comporte comme le bouton de soumission du formulaire et ne prend pas de paramètres.

Syntaxe : `form.submit()`

Exemple :

```
<form id="FORM1" method="post" action="/code.php">
<label>Nom <input type="text" name="nom"></label><br>
<label>Age <input type="text" name="Age"><label> <br>
<input type="submit" onclick="SoumettreForm()" value="SUBMIT">
<input type="button" onclick="ResetForm()" value="RESET">
</form>
<p id="message"></p>
<script>
  function SubmitForm() {
    document.getElementById("FORM1").submit();
  }
  function ResetForm() {
    document.getElementById("FORM1").reset();
    document.getElementById("message").innerHTML="Formulaire réinitialisé";
  }
</script>
```

CHAPITRE 2

Gérer les éléments d'un formulaire

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



02 - Gérer les éléments d'un formulaire

Interruption d'un formulaire



Empêcher la soumission d'un formulaire

La fonction `preventDefault()` empêche l'exécution de l'action par défaut de l'événement.

Exemple :

```
<form onsubmit="EnvoyerForm(event)">
  <input type="text">
  <button type="submit">Envoyer</button>
</form>

<script type="text/JavaScript">
  function EnvoyerForm(event){
    event.preventDefault();
    window.history.back();
  }
</script>
```

La méthode `window.history.back()` renvoie à l'URL précédente dans l'historique.

CHAPITRE 2

Gérer les éléments d'un formulaire

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



02 - Gérer les éléments d'un formulaire

Validation d'un formulaire



La validation des données

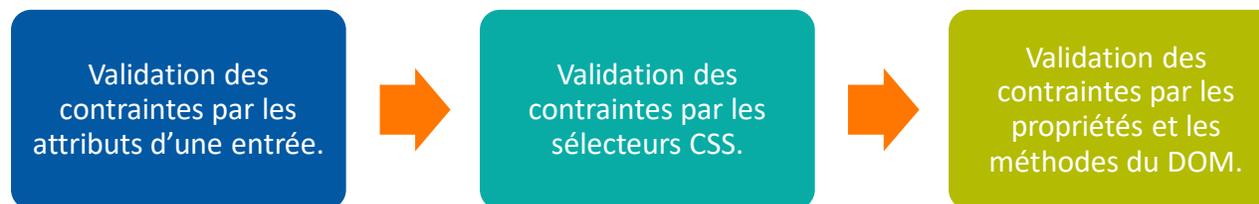
La validation des données consiste à s'assurer que l'entrée de l'utilisateur est conforme aux données attendues. Parmi les types de vérifications à faire on cite :

- l'utilisateur a-t-il rempli tous les champs obligatoires ?
- l'utilisateur a-t-il saisi une date valide ?
- l'utilisateur a-t-il saisi du texte dans un champ numérique ?
-

On distingue deux types de validation :

- **La validation côté serveur** : effectuée par un serveur Web, une fois que les données sont envoyées au serveur.
- **La validation côté client** : effectuée par un navigateur Web, avant que les données ne soient envoyées à un serveur Web.

HTML5 a introduit un nouveau concept de validation HTML appelé « validation des contraintes » qui est basée sur :



02 - Gérer les éléments d'un formulaire

Validation d'un formulaire



Validation des contraintes HTML

Validation des contraintes en utilisant les sélecteurs CSS

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

Validation des contraintes par les attributs

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 d'un formulaire

Exemple : Si un champ de formulaire (nom) est vide, la fonction affiche un message et renvoie false pour empêcher la soumission du formulaire

```
function validerForm() {  
    let x = document.forms["myForm"]["nom"].value;  
    if (x == "") {  
        alert("Le champ "nom" doit être saisi");  
        return false;  
    }  
}
```

```
<form name="myForm" action="/code.php" onsubmit="return validerForm()" method="post">  
    Nom: <input type="text" name="nom">  
    <input type="submit" value="Submit">  
</form>
```

02 - Gérer les éléments 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 (nom) est vide, l'attribut `required` empêche la soumission du formulaire

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



WEBFORCE
BE THE CHANGE



PARTIE 5

Manipuler JQUERY

Dans ce module, vous allez :

- Découvrir JQUERY
- Découvrir AJAX



18 heures

CHAPITRE 1

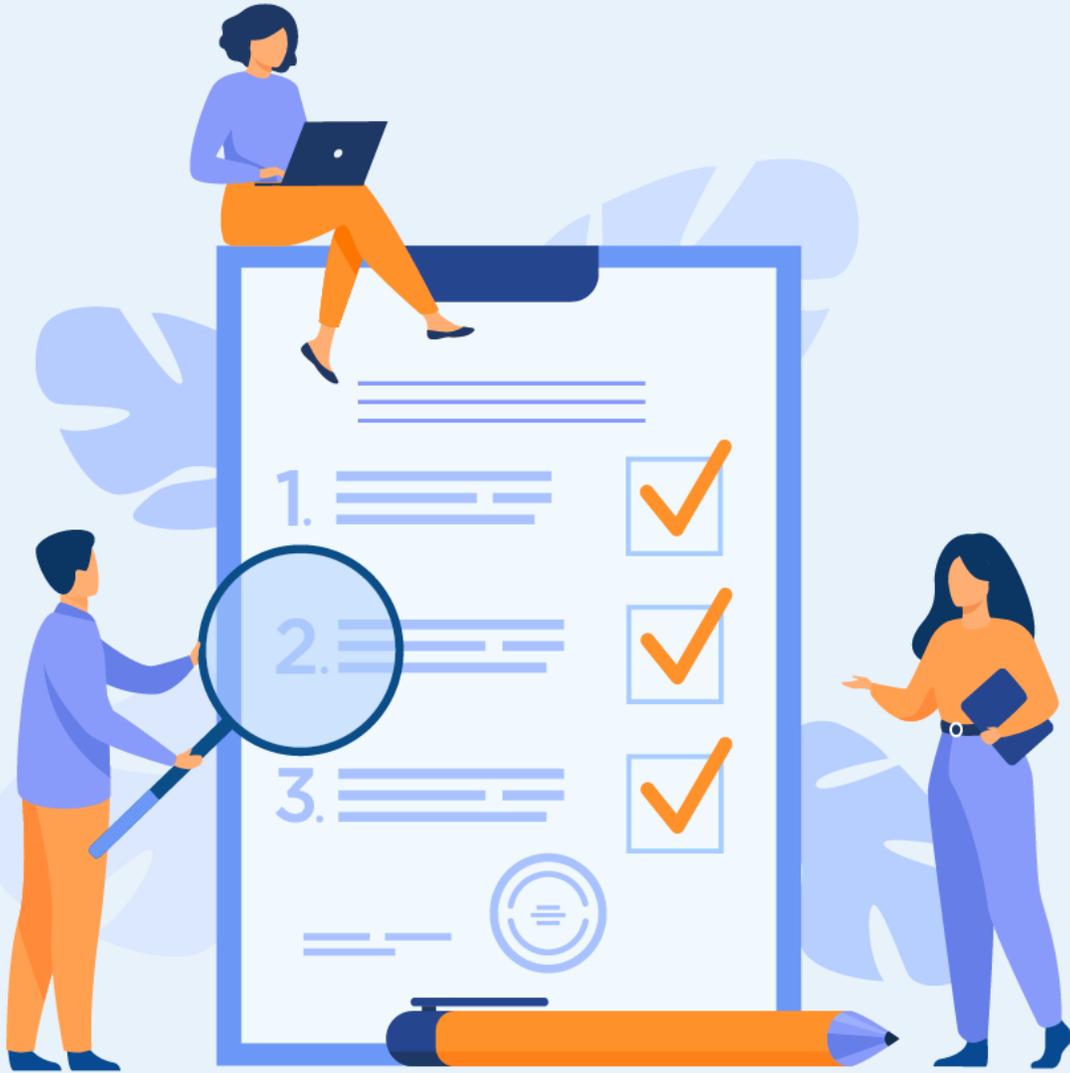
Découvrir JQUERY

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

JQUERY : introduction

JQuery est une bibliothèque JavaScript open-source inventée par John Resig en 2006. Cette bibliothèque est compatible avec les différents navigateurs web.

Le rôle de JQuery est de simplifier l'utilisation de JavaScript et la manipulation du DOM sur les site Web. En effet, les traitements nécessitant l'écriture de nombreuses lignes de code JavaScript sont encapsulés dans des méthodes appelées dans une seule ligne de code.

La bibliothèque jQuery contient les fonctionnalités suivantes :

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

01 - Découvrir JQUERY

Fonctions essentielles et chaînage



JQUERY : Installation

On peut utiliser deux manières pour utiliser JQuery dans les pages html :

Méthode 1 : Téléchargement de JQuery

Il existe deux versions de JQuery téléchargées depuis le site jquery.com.

- **Version de production** : version minifiée et compressée pour la phase de l'hébergement.
- **Version de développement** : version non compressée et lisible, pour la phase de développement et de tests.

La bibliothèque jquery téléchargée correspond à un fichier JavaScript. Pour l'utiliser il faut le référencer avec la balise <script> dans la section <head> :

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

Méthode 2 : JQuery via CDN (Content Delivery Network)

On peut inclure JQuery à partir d'un CDN (Content Delivery Network) sans besoin de télécharger les fichiers.

Exemple d'utilisation de JQuery hébergé par Google :

```
<head>  
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>  
</head>
```

01 - Découvrir JQUERY

Fonctions essentielles et chaînage



Syntaxe de JQUERY

La syntaxe de JQuery est basée sur les sélecteurs :

`$(sélecteur). action ()`

- **\$()** est un raccourci vers la fonction **jQuery()** qui trouve des éléments dans une page et crée des objets jQuery qui référencent ces éléments.

Par exemple : `$('p')` et `jQuery('p')` : sélectionne tous les éléments p (paragraphe).

- **Sélecteur** correspond à sélecteur CSS pour interroger (ou rechercher) des éléments HTML.
- **Action** correspond à une action à effectuer sur le(s) élément(s) sélectionnés.

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

Les méthodes jQuery se trouvent dans l'événement **ready** de l'objet document qui permet d'empêcher l'exécution du code jQuery avant la fin du chargement du document.

```
$(document).ready(function(){  
  
    // Méthodes jQuery...  
});
```

Exemples d'actions qui peuvent échouer si les méthodes sont exécutées avant le chargement complet du document :

- Masquer un élément qui n'est pas encore créé.
- Obtenir la taille d'une image qui n'est pas encore chargée.

Notion de chaînage

Les instructions **JQuery** peuvent être écrites soit l'une après l'autre (dans des lignes différentes) ou en utilisant la technique de chaînage.

Le chaînage permet d'exécuter plusieurs actions JQuery l'une après l'autre (dans la même ligne), sur le même élément.

Exemple : Les méthodes `css()`, `slideUp()` et `slideDown()` sont appelées sur le paragraphe identifié par l'ID « p1 ». Ainsi, l'élément "p1" devient d'abord rouge, puis il glisse vers le haut, puis vers le bas :

Syntaxe 1 :

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
```

Syntaxe 2 :

```
$("#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



Désactiver un lien href en JQUERY

La méthode `removeAttr()` de JQuery permet de supprimer l'attribut « href » du lien, qui devient non cliquable.

```
<head>
  <meta charset="utf-8">
  <title>Désactiver un lien</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.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://www.ofppt.ma/">Lien1</a></li>
    <li><a href="https://www.ofppt.ma/">Lien2</a></li>
    <li><a href="https://www.ofppt.ma/" class="disabled">Lien3</a></li>
  </ul>
</body>
```

Activer un lien href en JQUERY

La méthode `attr()` de JQuery permet d'ajouter l'attribut « href » à un lien.

```
<head>
  <meta charset="utf-8">
  <title>Désactiver un lien</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
  <script type="text/javascript">
    $(document).ready(function(){
      $(".menu a").each(function(){
        if($(this).hasClass("disabled")){
          $(this).attr("href","https://www.google.com/");
        }
      });
    });
  </script>
</head>
```

```
<body>
  <ul class="menu">
    <li><a href="https://www.ofppt.ma/">Lien1</a></li>
    <li><a href="https://www.ofppt.ma/">Lien2</a></li>
    <li><a class="disabled">Lien3</a></li>
  </ul>
</body>
```

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

Les événements en JQUERY

Les événements DOM ont une méthode jQuery équivalente.

Exemple 1 : La méthode `click()`

Attribuer un événement de clic et une fonction à tous les paragraphes d'une page. La fonction masque l'élément cliqué.

```
$("#p").click(function(){  
    // actions de l'évènement  
    $(this).hide();  
});
```

Exemple 2 : La méthode `dblclick()`

Attribuer un événement de double clic et une fonction à tous les paragraphes d'une page. La fonction est exécutée lorsque l'utilisateur double-clique sur le paragraphe.

```
$("#p").dblclick(function(){  
    $(this).hide();  
});
```

Les événements en JQUERY

Exemple 3 : La méthode `mouseenter()`

```
$("#p1").mouseenter(function(){  
    alert("Vous êtes sur le paragraphe p1!");  
});
```

Exemple 4 : La méthode `mouseleave()`

```
$("#p1").mouseleave(function(){  
    alert("vous avez quitté le paragraphe 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

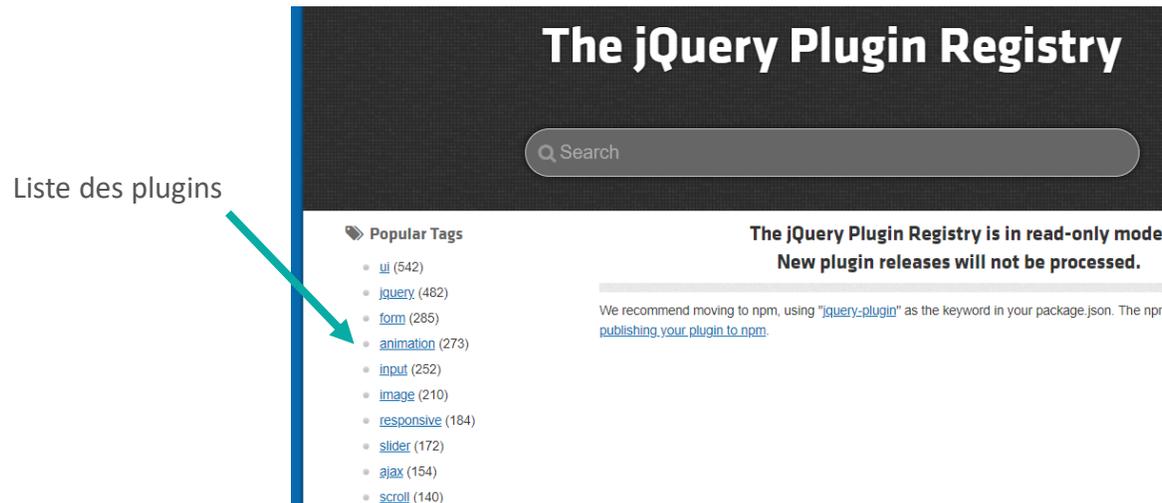


Intérêt des plugins

- Un plugin est un code écrit dans un fichier JavaScript standard. Il fournit des méthodes jQuery utiles qui peuvent être utilisées avec les méthodes de la bibliothèque jQuery.
- L'installation des modules JQuery additionnels, appelés **JQuery plugins** permet d'étendre les fonctionnalités offertes par JQuery et gagner en termes de rapidité du développement en réutilisant des codes existants.

Intégrer des plugins existants

- Il existe de nombreux sites proposant des plugins jQuery. Parmi ces sites, on peut utiliser le site officiel <https://jquery.com/plugins>.
- Pour intégrer un plugin dans une page HTML, il faut télécharger le plugin à partir du site dédié puis le référencer dans la page HTML.



Intégrer le plugin jquery.plugin-in.js téléchargé dans le fichier HTML :

```
<script src = "jquery.plugin-in.js" type =  
"text/javascript"></script>
```

Figure 26 : téléchargement des plugins JQuery

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

Utilisation de plugins existants

Utilisation de plugins existants

Exemple : <https://github.com/loebi-ch/jquery-clock-timepicker/blob/master/README.md>

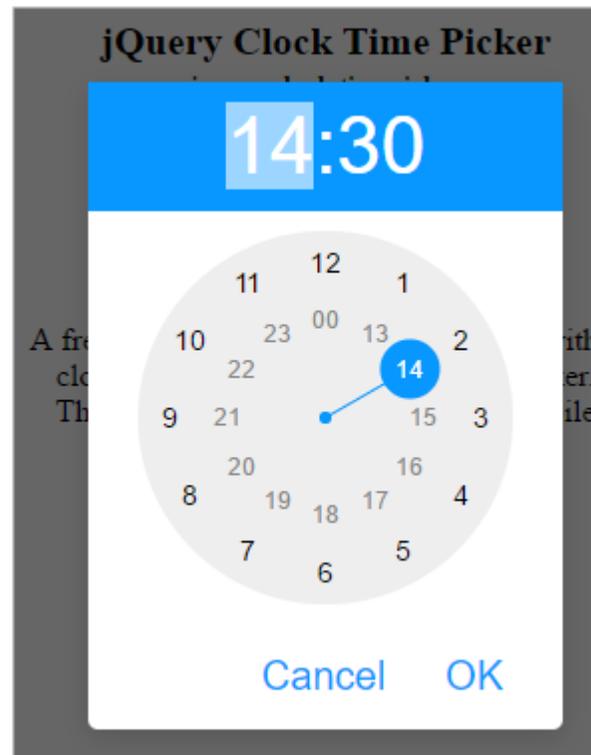
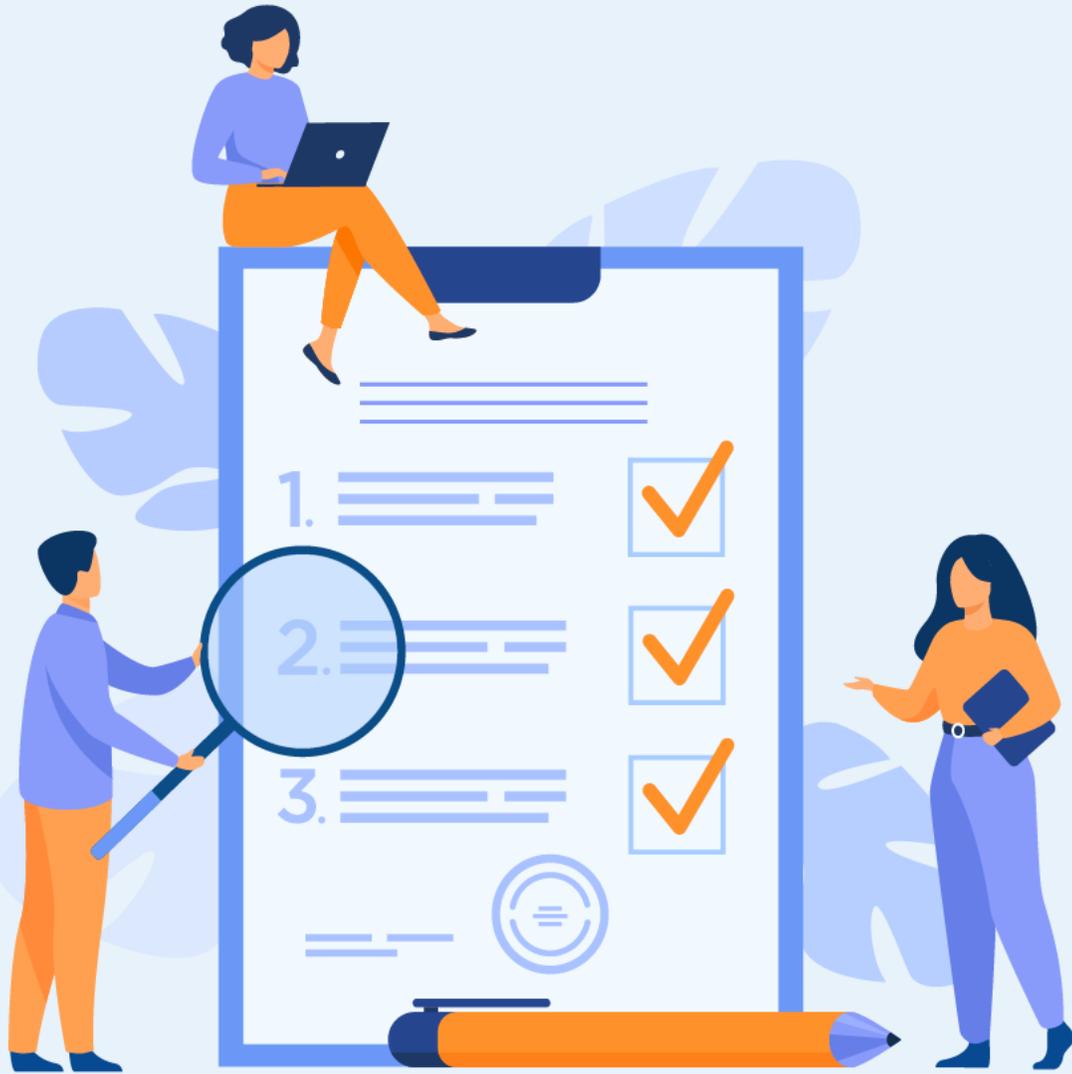


Figure 27: JQuery clock Time Picker [<https://github.com/loebi-ch/jquery-clock-timepicker>]



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



Qu'est-ce qu'AJAX ?

AJAX est acronyme de « Asynchronous JavaScript And XML ».

AJAX est une technologie basée sur :

- Un objet **XMLHttpRequest** intégré au navigateur (pour demander des données à un serveur Web).
- **JavaScript** et **DOM HTML** (pour afficher les données).

AJAX permet de :

- Lire les données d'un serveur web (après le chargement d'une page web) ;
- Mettre à jour une page web sans la recharger ;
- Envoyer les données à un serveur web (en arrière-plan).

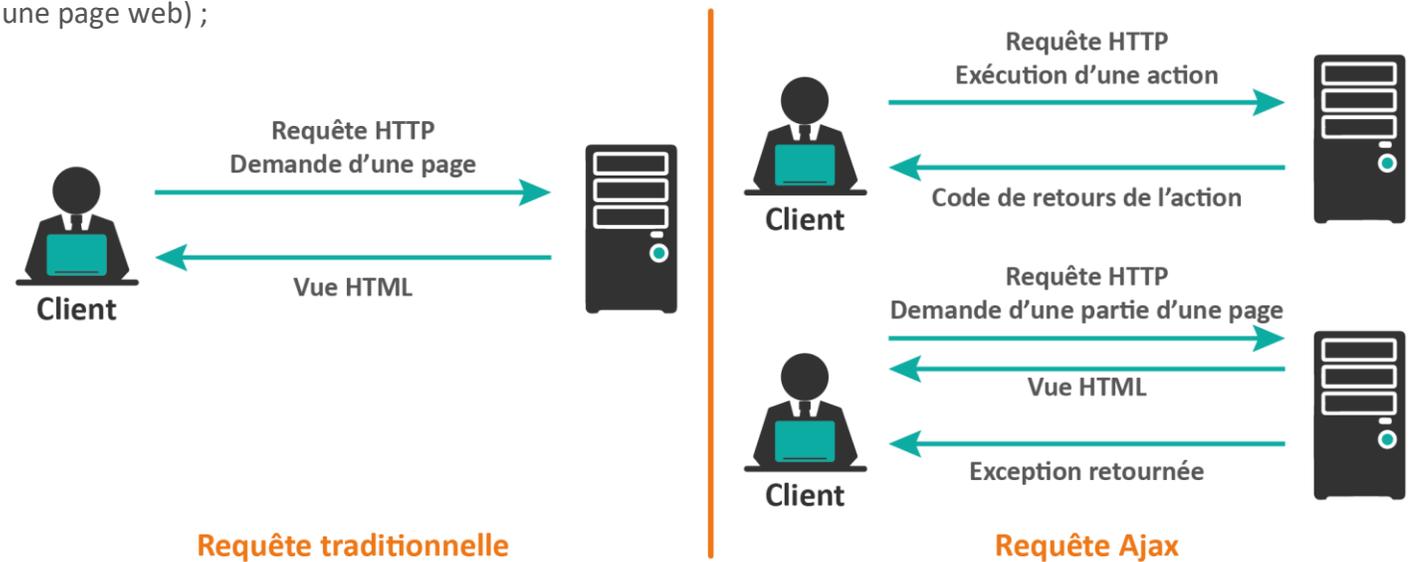


Figure 28 : Fonctionnement AJAX

L'objet XMLHttpRequest

- L'objet **XMLHttpRequest** de la technologie AJAX est un objet qui permet d'envoyer des requêtes HTTP au serveur, de recevoir des réponses et de mettre à jour 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.
- L'objet **XMLHttpRequest** est basé sur le principe d'échange de données entre le client (la page web) et le serveur (sur lequel se trouve la page ou la source de données à laquelle la page Web veut accéder).

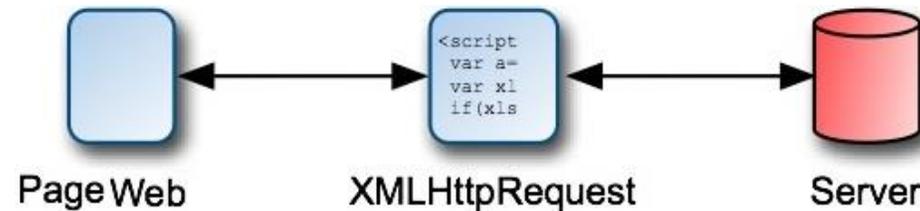


Figure 29 : l'objet XMLHttpRequest de AJAX

CHAPITRE 2

Découvrir AJAX

1. Introduction à AJAX
2. **Fonctionnement d'AJAX**
3. Implémentation d'AJAX via jQuery



Fonctionnement de l'objet XMLHttpRequest

L'objet **XMLHttpRequest (XHR)** est créé par le moteur JavaScript du navigateur pour initier des requêtes (demandes) HTTP à partir de l'application vers le serveur, qui à son tour renvoie la réponse au navigateur.

La propriété **XMLHttpRequest.readyState** renvoie l'état d'un client XMLHttpRequest. Les états possible du "readyState" sont :

- **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, les en-têtes de demande peuvent être définies 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)**.

Les requêtes du serveur doivent être envoyées de manière **asynchrone**: JavaScript n'a pas à attendre la réponse du serveur, et peut exécuter d'autres scripts en attendant la réponse du serveur ou bien traiter la réponse une fois que la réponse est prête.

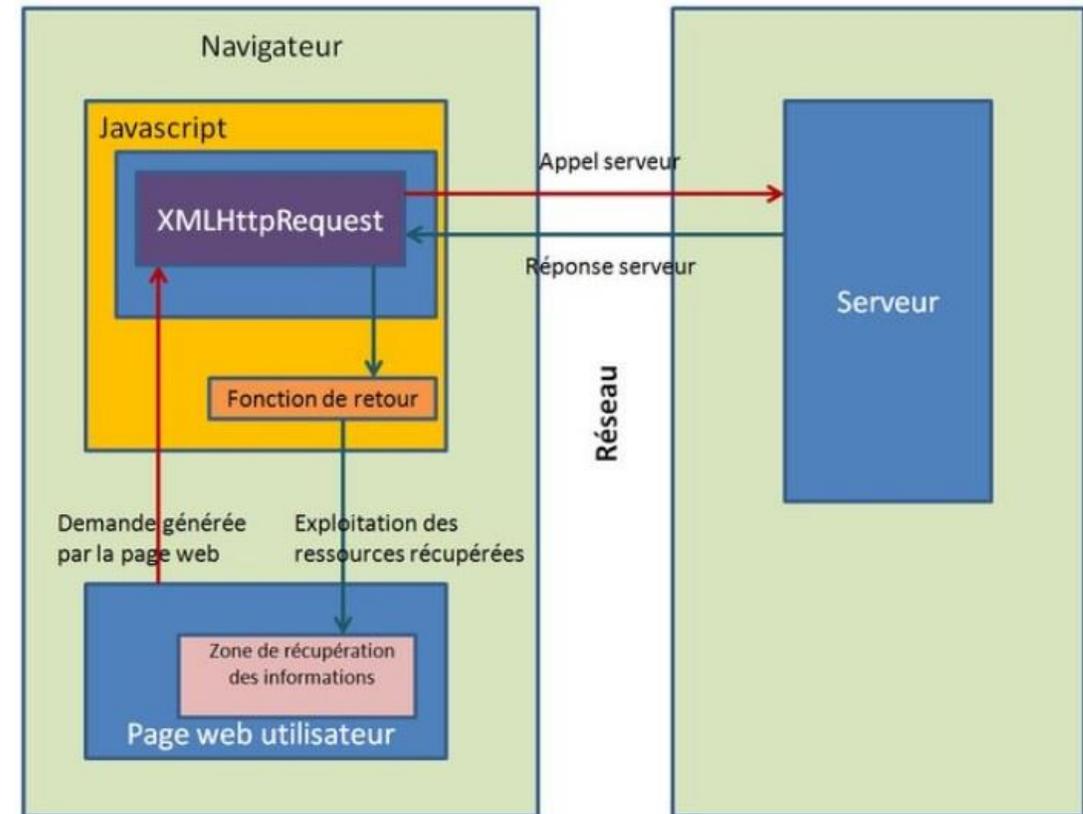


Figure 30 : [Source : <https://apcpedagogie.com/lobjet-xmlhttprequest/>]

Créer un objet XMLHttpRequest

Syntaxe de création d'un objet `XMLHttpRequest` (reconnu par la plupart des navigateurs)

```
variable = new XMLHttpRequest();
```

Méthodes d'objet XMLHttpRequest

Méthode	Description
<code>new XMLHttpRequest()</code>	Créer un nouvel objet XMLHttpRequest
<code>abort()</code>	Quitter la requête courante
<code>getAllResponseHeaders()</code>	Retourner toutes les informations du header
<code>getResponseHeader()</code>	Retourner une information spécifique du header
<code>open(method,url,async,user,psw)</code>	Spécifier la requête : <ul style="list-style-type: none">• method: GET ou POST• url: emplacement du fichier• async: true (asynchrone, c'est à dire JavaScript n'a pas à attendre la réponse du serveur) ou false (synchrone)• user: nom d'utilisateur (optionnel)• psw: mot de passe (optionnel)
<code>send()</code>	Envoyer la requête au serveur (utilisé dans les requêtes GET)
<code>send(string)</code>	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
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

Les méthodes `open()` et `send()` de l'objet XMLHttpRequest permettent d'envoyer une requête à un serveur

Syntaxe :

```
open(method, url, async)
```

Exemple 1 :

```
let xhttp = new XMLHttpRequest();  
xhttp.open("GET", "code.php", true);  
xhttp.send();
```

Exemple 2 : Envoi des données dans la requête GET

```
let xhttp = new XMLHttpRequest();  
xhttp.open("GET", "code.php?prenom=Hassan&nom=FILALI", true);  
xhttp.send();
```

Exemple 3 :

```
let xhttp = new XMLHttpRequest();  
xhttp.open("POST", "code.php", true);  
xhttp.setRequestHeader("Content-type", "application/x-www-form-  
urlencoded");  
xhttp.send("prenom=Hassan&nom=FILALI");
```

Pour utiliser la méthode dans un formulaire, il faut ajouter un en-tête HTTP avec `setRequestHeader()`

La propriété `onreadystatechange`

- La propriété `onreadystatechange` de l'objet `XMLHttpRequest` permet de définir une fonction à exécuter quand une réponse est reçue.
- La propriété `onreadystatechange` exécute la fonction chaque fois que `readyState` change de valeur : Lorsque `readyState` est 4 et `status` est 200, la réponse est prête.

Exemple 1 :

Source : <https://www.w3schools.com>

```
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();
```

Exemple 2 :

Source : <https://www.w3schools.com>

```
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.
- Les fonctions de rappel sont utilisées quand plusieurs tâches AJAX doivent être exécutées dans un site web : il faut créer une fonction pour exécuter l'objet **XMLHttpRequest** et une fonction de rappel pour chaque tâche AJAX.
- L'appel de la fonction doit contenir l'URL et la fonction à appeler lorsque la réponse est prête.

Exemple 1 :

Source : <https://www.w3schools.com>

```
<div id="demo">
<h1>Objet XMLHttpRequest</h1>
<button type="button"
onclick="loadDoc('ajax_info.txt', myFunction)">
Change Content
</button>
</div>
<script>
</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**



02 - Découvrir AJAX

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]);
```

URL : chaîne de l'URL vers laquelle les données sont soumises (ou récupérées).

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.

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Les options de 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"
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 attendues 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.
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 : 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.
password	Un mot de passe à utiliser avec XMLHttpRequest en réponse à une demande d'authentification d'accès HTTP.

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Envoyer une demande Ajax

La méthode ajax() effectue exécute une requête HTTP asynchrone et récupère les données du serveur.

Exemple :

```
$.ajax('/jquery/getdata',  
  {  
    success: function (data, status, xhr) {  
      $('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 on veut 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 qui spécifie la fonction de rappel qui sera exécutée.

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



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 des données de la réponse  
  timeout: 500,     // délai d'expiration en millisecondes  
  success: function (data,status,xhr) { // fonction callback en cas de succès  
    $('p').append(data.firstName + ' ' + data.middleName + ' ' + data.lastName);  
  },  
  error: function (jqXHR, textStatus, errorMessage) { // cas d'erreur  
    $('p').append('Error: ' + errorMessage);  
  }  
});  
<p></p>
```

Le premier paramètre est une URL de la requête qui renvoie des données au format JSON.

Dans le paramètre options, l'option **dataType** spécifie le type de données de réponse (JSON dans ce cas). L'option **timeout** spécifie le délai d'expiration de la demande en millisecondes.

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Envoyer une demande Ajax

Syntaxe équivalente à l'exemple précédent :

```
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>
```

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Envoyer une requête HTTP POST en utilisant ajax()

La méthode `ajax()` peut envoyer tout type de requêtes HTTP (GET, PSOT, PUT).

Exemple : Envoie d'une requête HTTP POST au serveur.

```
$.ajax('/jquery/submitData', {  
  type: 'POST', // Méthode POST  
  data: { myData: 'Mes données.' }, // données à envoyer  
  success: function (data, status, xhr) {  
    $('p').append('status: ' + status + ', data: ' + data);  
  },  
  error: function (jqXHR, textStatus, errorMessage) {  
    $('p').append('Error' + errorMessage);  
  }  
});  
<p></p>
```

Le premier paramètre est une URL de la requête qui renvoie des données au format JSON.

L'option **type** désigne le type de la requête (POST dans ce cas). L'option **data** spécifie que les données qui seront soumises au serveur le seront en tant qu'objet JSON.

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Méthode jQuery get()

La méthode jQuery `get()` envoie une requête http GET asynchrone au serveur et récupère les données.

Syntaxe :

```
$.get(url, [données],[rappel]);
```

URL : url de la requête à partir de laquelle on récupère les données.

Data : données à envoyer au serveur.

Callback : fonction à exécuter lorsque la requête aboutit.

Exemple : Récupération des données à partir d'un fichier texte.

```
$.get('/data.txt', // url
    function (data, textStatus, jqXHR) { // success callback
        alert('status: ' + textStatus + ', data: ' + data);
    });
```

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Méthode jQuery `getJSON()`

La méthode jQuery `getJSON()` envoie une requête http GET asynchrone au serveur et récupère les données au format JSON uniquement.

Rappel : La méthode `get()` récupère tout type de données.

Syntaxe :

```
$.getJSON(url, [données],[rappel]);
```

URL : url de la requête à partir de laquelle on récupère les données.

Data : données à envoyer au serveur.

Callback : fonction à exécuter lorsque la requête aboutit.

Exemple 1: Récupérer des données JSON

```
$.getJSON('/jquery/getjsondata', {name:'Ali'},  
function (data, textStatus, jqXHR){  
    $('p').append(data.firstName);  
});  
  
<p></p>
```

Exemple 2: Utiliser les méthodes de rappel `fail()` et `done()`

```
$.getJSON('/jquery/getjsondata', { name:'Ali'},  
function(data, textStatus, jqXHR){  
    alert(data.firstName);  
})  
.done(function () { alert('Request done!'); })  
.fail(function (jqxhr, settings, ex) {  
    alert('failed, '+ ex); });
```

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Méthode jQuery post()

La méthode jQuery **post()** envoie une requête HTTP POST asynchrone au serveur.

Syntaxe :

```
$.post(url, [données], [rappel], [type]);
```

URL : url de la requête à partir de laquelle on récupère / soumet les données.

Data : données JSON à envoyer au serveur.

Callback : fonction à exécuter lorsque la requête aboutit.

Type : type de données du contenu de la réponse.

Exemple 1 :

```
$.post('/jquery/submitData', // url
      { myData: 'This is my data.' }, // données à soumettre
      function(data, status, jqXHR) { // succès
          $('p').append('status: ' + status + ', data: ' + data);
      })
```

<p></p>

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Méthode jQuery post()

Exemple 2 : Les données JSON sont obtenues en tant que réponse du serveur. Ainsi, la méthode post() analysera automatiquement la réponse dans l'objet JSON.

```
$.post('/submitJSONData', // url
      { myData: 'This is my data.' }, // data to be submit
      function(data, status, xhr) { // succès
          alert('status: ' + status + ', data: ' + data.responseData);
      },
      'json'); // format des données de réponse
```

Exemple 3 : Attacher les méthodes de rappel fail() et done() à la méthode post().

```
$.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>

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



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]);
```

URL : url de la requête à partir de laquelle on récupère le contenu.

Data : données JSON à envoyer au serveur.

Callback : fonction à exécuter lorsque la requête aboutit.

Exemple : charger du contenu html depuis le serveur et l'ajouter à l'élément div.

```
$('#msgDiv').load('/demo.html');  
  
<div id="msgDiv"></div>
```



Remarque

- Si aucun élément n'est trouvé par le sélecteur alors la requête Ajax ne sera pas envoyée.

02 - Découvrir AJAX

Implémentation d'AJAX via jQuery



Méthode jQuery load()

La méthode **load()** permet de spécifier une partie du document de réponse à insérer dans l'élément DOM.

Ceci peut être réalisé à l'aide du paramètre **url**, en spécifiant un sélecteur avec une URL séparée par un ou plusieurs caractères d'espace.

Exemple : Le contenu de l'élément dont l'ID est **myHtmlContent** sera ajouté à l'élément **msgDiv**.

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
</head>
<body>
  <h1>Page html.</h1>
  <div id="myHtmlContent">This is my
html content.</div>
</body>
</html>
```

```
$('#msgDiv').load('/demo.html #myHtmlContent');

<div id="msgDiv"></div>
```