

## TABLE DES MATIERES

Chapitre 1 : Introduction .....	4
Chapitre 2 : Bases du HTML .....	5
Structure minimale d'un document HTML .....	5
L'élément <!DOCTYPE> .....	5
L'élément <html> .....	6
L'élément <head> .....	6
L'élément <body> .....	7
Les commentaires en HTML .....	7
La validation .....	7
Attributs globaux en HTML .....	8
Quelques terminologies .....	8
Chapitre 3 : Eléments de texte .....	10
Paragraphes .....	10
En-têtes .....	10
Sauts .....	10
Listes .....	11
Listes Non Ordonnées .....	11
Listes Ordonnées .....	11
Listes de définition .....	11
Listes imbriquées .....	12
Eléments de texte divers .....	12
Eléments de texte en ligne .....	12
Texte Emphatique .....	13
Texte Fort ou Important .....	13
Texte en Italique .....	13
Texte en Gras .....	13
Texte Surligné .....	13
Texte de Petite Taille .....	13
Indices et Exposants .....	13
Texte de Code .....	13
Citations Courtes .....	14
Abréviations .....	14
Date et Heure .....	14
Eléments génériques .....	14
Élément générique de Bloc .....	14
Élément générique en Ligne .....	15
Eléments d'organisation du document .....	15
Séquences d'échappement .....	15
Chapitre 4 : Liens hypertextes .....	15
Chapitre 5 : Images .....	16
Formats d'images pour le web .....	17

L'élément <img/> .....	17
Images responsives .....	18
Découverte des attributs de <source/> .....	19
Dessiner des formes SVG .....	19
Chapitre 6 : Tableaux .....	19
Structure minimale d'un tableau .....	19
Etendre les cellules d'un tableau .....	20
Les éléments de groupage .....	21
Chapitre 7 : Formulaires HTML .....	22
Comment les formulaires fonctionnent-ils ? .....	22
L'élément <form> .....	22
Les variables de formulaire .....	23
L'élément <textarea> .....	24
L'élément <input/> .....	25
input type="button" .....	25
input type="submit" .....	26
input type="image" .....	27
input type="reset" .....	27
input type="checkbox" .....	27
input type="radio" .....	28
input type="date" .....	28
input type="time" .....	29
input type="datetime-local" .....	29
input type="month" .....	30
input type="week" .....	31
input type="color" .....	31
input type="range" .....	32
input type="file" .....	32
input type="text" .....	33
input type="email" .....	33
input type="tel" .....	34
input type="search" .....	34
input type="url" .....	35
input type="password" .....	36
input type="number" .....	36
input type="hidden" .....	37
L'élément <datalist> .....	37
L'élément <select> .....	38
L'élément <button> .....	38
L'élément <output> .....	39
L'élément <progress> .....	39
L'élément <meter> .....	40

Les éléments d'accessibilité pour les formulaires .....	40
L'élément <label> .....	40
L'élément <fieldset>.....	40
Chapitre 8 : Autres Elements multimedia .....	41
L'élément <iframe> .....	41
L'élément <object> .....	42
L'élément <audio> .....	43
L'élément <video> .....	44
L'élément <canvas> .....	42

## CHAPITRE 1 : INTRODUCTION

Internet fonctionne en respectant les normes du protocole TCP/IP. Ce protocole a cinq couches (une couche est une séparation imaginaire qui catégorise les opérations qu'on effectue pour réaliser un objectif) : La couche physique, la couche de liaison de données, la couche réseau, la couche de transport et la couche application. Lorsqu'un ordinateur envoie des données à un autre à travers internet, les données dans l'ordinateur qui envoient part de la couche application à la couche physique et l'ordinateur qui les reçoit les récupère de la couche physique et les retraduit jusqu'à la couche application.

1. La **couche physique** est implémentée lorsqu'on connaît les caractéristiques des moyens physiques qu'on va utiliser pour échanger les données. Ex : la nature de la communication (sans fil, par fibre, etc.) et les détails physiques associés (longueur d'onde, distance, etc.).
2. La **couche de liaison de données** est matérialisée par les supports de transmission utilisés pour faire voyager les données. Généralement, un ordinateur communique sur internet grâce à des proxys. Un proxy est tout intermédiaire entre l'ordinateur qui envoie et celui qui reçoit. De manière grossière, pour communiquer sur internet, il doit y avoir au moins un proxy entre l'ordinateur émetteur et l'ordinateur récepteur : le fournisseur d'accès à internet (ex : Vodacom, CanalBox, etc.). La couche de liaison des données est implémentée lorsque l'ordinateur émetteur est physiquement connecté à l'ordinateur récepteur. Chaque ordinateur est connecté à un FAI par fibre optique ou par antenne et chaque FAI est connecté à l'autre par fibre optique ou par satellite.
3. La **couche réseau** est implémentée lorsque l'ordinateur émetteur est connecté à l'ordinateur récepteur grâce à un routeur ou connecté au FAI à travers un modem. Ces derniers leur donnent chacun une adresse IP unique, un identifiant unique sur internet crucial pour que les deux machines sachent où ils se trouvent et quelles routes prendre pour communiquer.
4. La **couche de transport** est implémentée lorsque le système d'exploitation respecte les règles précises de transport pour traiter les données venant ou allant vers la carte réseau. Par exemple, la règle TCP qui dit que l'envoi ou la réception d'une donnée doit se passer pendant une session où on établit la connexion en ouvrant un port TCP, on transfère ou on reçoit les données puis on ferme le port. Il y a aussi l'UDP, le RTP, etc. De base, cette couche est implémentée par le système d'exploitation.
5. La **couche application** est implémentée lorsqu'on a dans son ordinateur des logiciels créés en respectant les règles de cette couche pour envoyer et recevoir des données via internet (client HTTP, serveurs HTTP, clients DNS, serveurs DNS, clients SSH, etc.).

Le web fonctionne grâce aux applications qui respectent la norme HTTP. HTTP est un ensemble de règles qui dictent comment les applications qui veulent échanger des données hypertexte via internet doivent être implémentées. Ces applications doivent envoyer des commandes au système d'exploitation. Celui-ci se charge alors de les envoyer là où il faut en suivant à son tour le protocole de transport TCP. Il y a 2 types de commandes HTTP : les **requêtes** et les **réponses**. Les requêtes HTTP sont émises par des clients (*il faut donc avoir un logiciel client HTTP pour écrire et émettre des commandes de requête HTTP et traiter les réponses*) et les réponses sont émises par des serveurs (*il faut aussi avoir un logiciel serveur HTTP pour traiter les requêtes et émettre des commandes de réponse HTTP*).

L'un des clients HTTP les plus populaires est le **navigateur web**. Il s'agit d'un logiciel dont le but principal est d'émettre des commandes de requête HTTP, d'interpréter les réponses et de les présenter à l'écran sous une forme agréable à interpréter par l'utilisateur.

Un **serveur HTTP** est lui, un logiciel qui attend des requêtes (elle écoute généralement sur le port TCP 80 en production, ou 30 en développement) et sur base du contenu de celles-ci, fournit des réponses HTTP. Etant donné que les serveurs sont idéalement attendus à ne jamais s'arrêter de fonctionner, ils sont installés dans des ordinateurs spéciaux appelés aussi **serveurs** qui sont bons pour fonctionner sans s'arrêter.

Supposons que vous avez demandé une page web, lorsque vous recevez la réponse du serveur pour la page que vous avez demandé, le navigateur lit le code HTML dans le corps de la réponse et utilise les éléments HTML qui s'y trouvent pour dessiner la page, renvoyer d'autres requêtes pour télécharger les données supplémentaires spécifiées dans le HTML comme les images, le CSS, le code JavaScript...

Les requêtes et réponses HTTP sont automatiquement créées et interprétées par le navigateur, on n'a pas besoin de taper toute leur syntaxe à chaque fois qu'on veut visiter une page.

Le seul control que nous avons dans le HTTP est le contenu du corps de nos requêtes et réponses. Notre responsabilité est de décrire ce que nous voulons envoyer ou recevoir. Dans le cadre des pages Web, le premier élément à décrire est la structure, et pour se faire, on utilise un langage spécialement conçu pour décrire la structure d'une page Web : Le **HTML**. Il s'agit donc d'un langage qui permet de décrire au navigateur l'aspect sémantique d'une page Web (*de quoi est composé ma page et quelle est la hiérarchie des objets qui la composent ?*).

## CHAPITRE 2 : BASES DU HTML

L'unité fondamentale d'une page HTML est l'**élément**. Un élément décrit une portion de la page et lui attribue une signification sémantique (ex : l'élément `<p>` décrit que le texte qui s'y trouve est un paragraphe).

Un élément consiste essentiellement en une ou deux balises dans lequel peut optionnellement se trouver du contenu textuel ou multimédia :

- Les éléments à deux balises** : Ces éléments sont constitués d'une balise d'ouverture et d'une balise de fermeture. La balise d'ouverture peut contenir des attributs et détermine le début de l'élément, tandis que la balise de fermeture indique la fin de cet élément. La syntaxe est la suivante :

`<nomElement attributs>Contenu</nomElement>`

- Les éléments à une balise auto-fermant** : Ces éléments auto-fermant n'ont pas de balise de fermeture distincte. Ils sont utilisés pour intégrer des éléments simples comme des images, des liens, ou des métadonnées. La syntaxe de ces éléments peut être écrite sous la forme suivante :

ou    `<nomElement attributs />` (écriture stricte)  
                       `<nomElement attributs >` (écriture non-stricte)

Le contenu d'un élément HTML peut être remplaçable (comme le contenu d'un `<input>`) ou non-replaçable (comme le texte d'un `<p>` ou d'un `<div>`). Le contenu remplaçable permet à l'utilisateur d'interagir directement avec lui, tandis que le contenu non-replaçable est statique et affiché par le navigateur.

Enfin, le contenu de chaque élément HTML a sa manière par défaut d'être affiché. Il peut être affiché « en bloc » ou « en ligne ». Un élément en bloc, comme un `<div>` ou un `<h1>`, occupe toute la largeur disponible de son conteneur et commence sur une nouvelle ligne, tandis qu'un élément en ligne, comme un `<span>` ou un `<a>`, s'affiche sur la même ligne que les éléments adjacents et ne commence pas une nouvelle ligne. Cependant, cette propriété n'est pas à prendre en compte lorsqu'on écrit du HTML. Étant en rapport avec la présentation de la page et non l'aspect sémantique, elle peut être modifiée en CSS pour répondre aux besoins de mise en forme et de style de la page.

## STRUCTURE MINIMALE D'UN DOCUMENT HTML

On décrit une page ou document HTML dans un fichier texte de format `.html`. Voici la structure minimale :

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8" name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Syntaxe minimale d'une page HTML</title>
</head>
<body>
  <!--Le corps de la page vient ici--&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>

```

### L'ELEMENT `<!DOCTYPE>`

L'élément `<!DOCTYPE>` est impératif dans certains documents de la famille XML. Dans un fichier HTML, il permet de spécifier le type du document ainsi que la version du HTML utilisé (dans ce cas, la version 5), indiquant ainsi comment

il doit être interprété par le navigateur. Dans la version 5 du HTML, `<!DOCTYPE>` ne s'emploie qu'une fois dans un document HTML, à la première ligne et n'accepte que l'attribut `html` et ce, obligatoirement.

## L'ELEMENT `<HTML>`

L'élément `<html>` est la balise de base qui enveloppe tout le contenu d'un document HTML. Elle est la racine de toute page web, et chaque élément HTML doit se trouver à l'intérieur de cette balise pour que le document soit correctement structuré et interprété par les navigateurs. Cette balise ne doit contenir que les deux sections suivantes : `<head>` et `<body>`.

## L'ELEMENT `<HEAD>`

L'élément `<head>` est une partie essentielle de tout document HTML. Elle contient des informations sur la page web, c'est-à-dire des données à propos du document qui ne sont pas affichées directement dans le navigateur, mais qui influencent le rendu, l'optimisation, et la structure de la page. Elle est toujours placée avant la balise `<body>` dans la structure d'un fichier HTML. Elle ne peut contenir que les 7 éléments suivants :

- `<title>` : Définit le titre qui apparaît dans l'onglet du navigateur et dans les résultats de recherche. Elle ne prend en charge qu'un contenu textuel et n'a pas d'attributs spécifiques.
- `<meta/>` : est utilisée dans la section `<head>` pour fournir des métadonnées sur la page web. Elle ne contient pas de contenu visible et n'a pas de balise de fermeture. Voici les principaux attributs que l'on peut utiliser avec `<meta/>` et leur utilité :
  - `charset` : Définit l'encodage de caractères de la page. En HTML5, cet encodage est souvent défini comme "UTF-8" pour supporter la plupart des caractères internationaux.
  - `name` : Décrit le nom de la métadonnée qu'on veut renseigner (ex : "author", "keywords", etc. Allez à <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/meta/name> pour la liste complète de tous les noms de métadonnées utilisables).
  - `http-equiv` : permet à la balise `<meta/>` de fonctionner comme une instruction HTTP.
  - `property` : permet d'indiquer le type de donnée Open Graph.

**Nota :** On ne peut utiliser qu'un seul des 3 attributs `name`, `property` et `http-equiv` dans un `<meta/>`. Si on veut en renseigner plusieurs, on doit déclarer chacun dans son `<meta/>`. `content` est un attribut spécial utilisé pour spécifier la valeur des informations fournies par `name`, `property` ou `http-equiv`. Il contient généralement l'information à transmettre au navigateur ou aux moteurs de recherche concernant la métadonnée à laquelle il est rattaché.

- Pour voir les valeurs qu'on peut affecter à `content` pour un `<meta/>` utilisant l'attribut `name`, visitez <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/meta/name>.
- Pour voir les valeurs qu'on peut affecter à `content` pour un `<meta/>` utilisant l'attribut `http-equiv`, <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/meta#http-equiv>.
- Pour celles qu'on peut affecter à `content` pour un `<meta/>` utilisant l'attribut `property`, voir <https://seosetups.com/blog/open-graph/>.
- `<link/>` : Sert à lier des ressources externes au document HTML. Elle est couramment utilisée dans la section `<head>` pour intégrer des feuilles de style CSS, des icônes, ou des polices d'écriture. Elle aussi ne contient pas de contenu visible et n'a pas de balise de fermeture. Voici les attributs les plus couramment utilisés avec lui :
  - `rel` (obligatoire) : Spécifie la relation entre le document HTML et la ressource liée. Les valeurs courantes sont "stylesheet" (Pour lier une feuille de style CSS externe), "icon" (Pour lier une icône (favicon) affichée dans l'onglet du navigateur), "preload" (Pour précharger des ressources (comme des polices ou des images) et améliorer les performances), "alternate" (Pour lier une version alternative de la page, comme une version imprimable ou une version en d'autres langues).
  - `href` (obligatoire) : Spécifie l'URL de la ressource à lier, comme un fichier CSS, une icône, etc.
  - `type` (facultatif) : Spécifie le type de contenu de la ressource liée (ex : "image/x-icon", "text/css" ...). Pour une liste exhaustive, voir <https://www.iana.org/assignments/media-types/media-types.xhtml>.
  - `media` (facultatif) : Permet de définir pour quels types d'appareils la page est destinée afin d'utiliser des styles adaptés. Voir [https://www.w3schools.com/tags/att\\_link\\_media.asp](https://www.w3schools.com/tags/att_link_media.asp).
- `<style>` : Est utilisée pour intégrer du CSS (Cascading Style Sheets) directement dans un document HTML. Contrairement à la balise `<link/>`, qui lie une feuille de style externe, la balise `<style>` contient des règles CSS définies à l'intérieur du document. Cette balise est généralement placée dans la section `<head>` du

- document. Bien qu'il n'ait pas beaucoup d'attributs, **media** est souvent utilisé et permet d'appliquer certains styles uniquement pour les écrans ou pour l'impression par exemple.
- **<script>** : Est utilisée pour inclure des scripts JavaScript dans un document HTML. Elle permet d'intégrer du code JavaScript directement dans le fichier HTML ou de lier des fichiers JavaScript externes. La balise **<script>** est généralement placée dans la section **<head>** ou à la fin de la section **<body>**, en fonction de la manière dont vous souhaitez charger et exécuter le script. Elle peut contenir plusieurs attributs importants :
    - **src** : Spécifie l'URL du fichier JavaScript externe à charger. Lorsque cet attribut est utilisé, le contenu de la balise **<script>** est ignoré.
    - **type** : Définit le type de contenu du script. La valeur par défaut est "**text/javascript**".
    - **async** : Indique que le script doit être téléchargé en parallèle avec le reste de la page et exécuté dès qu'il est prêt. L'exécution du script n'interrompt pas le rendu de la page.
    - **defer** : Indique que le script doit être exécuté après que le document HTML a été complètement analysé. Cela permet de s'assurer que le DOM est entièrement chargé avant l'exécution du script.
  - **<noscript>** : Est utilisée en HTML pour fournir un contenu alternatif destiné aux utilisateurs dont le navigateur ne prend pas en charge JavaScript ou lorsque JavaScript est désactivé dans leur navigateur. Elle permet de s'assurer que certaines informations ou fonctionnalités restent accessibles même si le JavaScript ne peut pas être exécuté. Elle peut contenir n'importe quel type de contenu HTML, comme du texte, des images, des liens, ou même d'autres éléments HTML.
  - **<base/>** : spécifie l'URL de base pour toutes les URL relatives contenues dans le document. Elle prend deux attributs :
    - **href** (obligatoire) : Définit l'URL de base.
    - **target** (facultatif) : Indique comment les liens doivent s'ouvrir. Les valeurs peuvent être "**\_self**" (ouvre les liens dans le même onglet (par défaut)), "**\_blank**" (ouvre les liens dans un nouvel onglet), "**\_parent**" (ouvre les liens dans le cadre parent), "**\_top**" (ouvre les liens dans une toute nouvelle fenêtre du navigateur).

(ex :

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Une introduction aux balises HTML essentielles" />
  <title>Introduction au HTML</title>
  <link rel="stylesheet" href="style.css" />
  <link rel="icon" href="favicon.ico" type="image/x-icon" />
  <meta property="og:title" content="Introduction au HTML" />
</head>
<body>
  <!--Le corps de la page vient ici-->
</body>
</html>.
```

## L'ELEMENT <BODY>

L'élément **<body>** est un élément essentiel dans un document HTML, car il contient tout le contenu visible de la page Web. Tout ce que les utilisateurs voient, comme le texte, les images, les vidéos, les liens, les formulaires, et d'autres éléments interactifs (en bref, tout ce qu'on apprendra dans les chapitres suivants), s'écrit dans cette balise.

---

## LES COMMENTAIRES EN HTML

L'élément **<!-- -->** est utilisé pour insérer des commentaires dans le code HTML. Ces commentaires ne sont pas visibles par les utilisateurs dans le navigateur, mais ils peuvent être utiles pour les développeurs afin d'ajouter des notes ou des explications dans le code (ex : **<!--Bonjour, Christian ! 😊-->**).

---

## LA VALIDATION

Lorsqu'un élément HTML est mal écrit, il n'y a pas de message d'erreur produit. Le navigateur l'ignore simplement. Cela peut rendre difficile le débogage d'un code HTML. Pour pallier à ce problème, on utilise un validateur. Un validateur HTML est un outil qui analyse votre code et signale les erreurs de syntaxe, les balises manquantes ou mal fermées, et d'autres problèmes potentiels.

En utilisant un validateur, comme le **W3C Markup Validation Service** (<https://validator.w3.org/>), les développeurs peuvent s'assurer que leur code respecte les normes HTML, ce qui améliore la compatibilité entre les différents navigateurs et rend la page plus accessible aux utilisateurs. En outre, un code HTML valide contribue à un meilleur référencement (SEO) et à une expérience utilisateur optimisée.

## ATTRIBUTS GLOBAUX EN HTML

Chaque élément a une liste d'attributs qu'il peut accepter. Cependant, il existe quelques attributs que tous les éléments en HTML acceptent. En voici quelques-uns :

- **class** : Cet attribut permet de spécifier une ou plusieurs classes CSS pour un élément, ce qui facilite la mise en forme et le ciblage d'éléments spécifiques via des styles CSS ou des scripts JavaScript (ex : `<div class="conteneur">Contenu ici</div>`).
- **id** : Fournit un identifiant unique à un élément dans le document. Cet identifiant peut être utilisé pour le ciblage CSS, le scripting JavaScript, ou pour créer des liens internes vers des sections spécifiques d'une page (ex : `<h1 id="titre-principal">Titre de la Page</h1>`).
- **lang** : Spécifie la langue du contenu de l'élément, ce qui est important pour l'accessibilité et le référencement. Il aide également les moteurs de recherche et les outils d'accessibilité à mieux comprendre le contenu (ex : `<p lang="fr">Ceci est un paragraphe en français.</p>`).
- **title** : Fournit des informations supplémentaires sur un élément. Lorsque l'utilisateur survole l'élément, une infobulle s'affiche avec le texte spécifié dans cet attribut. Cela peut être utile pour fournir des descriptions ou des conseils (ex : `<a href="#" title="En savoir +">Lien informatif</a>`).
- **draggable** : Spécifie si un élément peut être glissé ou non par l'utilisateur. Il est utilisé pour implémenter des fonctionnalités de « drag-and-drop ». Les valeurs possibles sont "true" (l'élément est glissable), "false" (l'élément ne peut pas être glissé) et "auto" (par défaut, le comportement est défini par l'élément et le navigateur) (ex : ``).
- **dropzone** : Spécifie les types de contenu qui peuvent être déposés dans l'élément. Les valeurs possibles sont "copy" (copie les données du contenu glissé dans l'élément), "move" (déplace les données du contenu) et "link" (crée un lien vers les données du contenu) (ex : `<div dropzone="copy">Déposez ici pour copier</div>`).
- **hidden** : Indique qu'un élément est masqué, sans être supprimé du DOM. Il n'a pas besoin de valeur ; simplement ajouter l'attribut masque l'élément (ex : `<p hidden>Paragraphe est caché.</p>`).
- **translate** : Spécifie si le contenu d'un élément doit être traduit ou non par les outils de traduction automatique. Les valeurs possibles sont "yes" (le contenu peut être traduit) et "no" (le contenu ne doit pas être traduit) (ex : `<p translate="no">Nom de marque</p>`).
- **inert** : Empêche un élément d'interagir avec l'utilisateur. On l'utilise surtout dans les applications de Single Page Application (SPA) pour désactiver certaines parties de la page (ex : `<div inert>Section désactivée</div>`).

En plus de ces attributs, il y a certains qui sont appelés « **attributs de gestion d'événements** ». Ils permettent d'exécuter une fonction (`"nomFonction()"`) ou un ou plusieurs instructions JavaScript en réaction à un quelconque changement détecté par le navigateur de la part d'un élément. (ex : `onclick`, `onblur`...).

Pour en savoir plus sur les attributs de gestion d'événements qui peuvent être utilisés sur n'importe quel élément HTML, voir [https://www.w3schools.com/tags/ref\\_eventattributes.asp](https://www.w3schools.com/tags/ref_eventattributes.asp).

## QUELQUES TERMINOLOGIES

Avant de chuter, quelques terminologies sont essentielles à connaître, notamment les termes « pages statiques simples », « pages statiques complexes », « pages dynamiques », « sites web statiques », « sites web dynamiques », et « applications web ».

Une **page statique simple** est une page HTML qui ne change pas en fonction des interactions de l'utilisateur. Elle est composée de code HTML, CSS, et parfois d'un peu de JavaScript pour des éléments basiques comme des animations ou des effets visuels, mais sans contenu généré dynamiquement. Par exemple, une page de présentation d'une entreprise avec des informations de contact et des images, sans besoin de mise à jour régulière ni d'interaction utilisateur.

Une **page statique complexe** est toujours construite en HTML, CSS, et JavaScript, mais elle intègre des éléments plus avancés, souvent rendus dynamiquement du côté client (par exemple, avec des frameworks comme React ou Vue, bien que la page elle-même reste statique). Ces pages peuvent inclure des animations, des effets de navigation, ou des composants interactifs côté client. Le contenu reste néanmoins figé, car il ne dépend pas d'une base de données ou d'un serveur pour fournir des données différentes. Un exemple pourrait être une page de galerie de photos avec des animations de défilement et de filtres interactifs.

Une **page dynamique** est générée à la demande par le serveur en fonction de l'interaction de l'utilisateur ou des données disponibles en arrière-plan. Ce type de page récupère des informations d'une base de données ou d'une API, et les présente en temps réel. Par exemple, une page de profil utilisateur qui affiche des informations spécifiques à l'utilisateur connecté, ou une page de blog qui affiche différents articles en fonction de ce que le serveur récupère.

Dans une page dynamique, des langages côté serveur comme PHP, Python, ou Node.js sont souvent utilisés pour construire la page en temps réel avant de l'envoyer au navigateur. À chaque nouvelle requête (ou à chaque actualisation de la page), le serveur génère un contenu actualisé.

Un **site web statique** est composé uniquement de pages statiques (simples ou complexes) qui sont toutes définies à l'avance et qui ne changent pas en fonction de l'utilisateur. L'ensemble du site peut être déployé sur un serveur de fichiers statique sans qu'aucun traitement dynamique ne soit nécessaire. Ce type de site est idéal pour des projets où le contenu est rarement modifié, comme des portfolios ou des sites d'information avec du contenu figé.

Avec l'émergence de générateurs de sites statiques (comme Jekyll ou Hugo), même des sites web au contenu riche peuvent être entièrement statiques en pré-générant les pages. Cela améliore la performance, car le serveur se contente de délivrer des fichiers déjà existants, sans logique dynamique.

Un **site web dynamique** contient des pages ou des sections qui changent en fonction des données ou des actions de l'utilisateur. Ces sites reposent sur une logique de traitement côté serveur pour fournir des informations qui changent en fonction des interactions ou des contextes. Les sites dynamiques peuvent intégrer des systèmes de gestion de contenu (CMS), des comptes utilisateurs, des forums, et tout autre contenu qui nécessite une mise à jour constante.

Les sites dynamiques utilisent des bases de données et du code côté serveur pour gérer et mettre à jour le contenu en temps réel, ce qui les rend plus interactifs et personnalisables.

Une **application web** va au-delà d'un site web dynamique en offrant des fonctionnalités et des interactions comparables à une application de bureau ou mobile. Elle exécute une logique métier complexe directement dans le navigateur et offre souvent des services interactifs en temps réel. Contrairement à un site web où le contenu est principalement informatif, une application web répond aux actions des utilisateurs de manière plus élaborée et prend en charge des processus métiers.

Les applications web se basent beaucoup sur JavaScript (avec des frameworks comme React, Vue, Angular) pour gérer des interfaces utilisateur avancées, et elles communiquent souvent de manière continue avec un serveur via des API pour échanger des données sans recharger la page. Des exemples d'applications web incluent les gestionnaires de tâches en ligne, les réseaux sociaux, ou les plateformes de messagerie.

Ouvrir des fichiers HTML directement avec un navigateur est possible parce qu'HTML est un langage de balisage interprété, conçu pour afficher du contenu de manière statique. Lorsqu'on double-clique sur un fichier .html, le navigateur lit le fichier et rend les éléments (textes, images, liens) directement. C'est suffisant pour des pages statiques simples, mais dès qu'il y a des interactions plus complexes (JavaScript, requêtes, etc.), on a besoin d'un serveur pour deux raisons essentielles : gestion des requêtes et interactions dynamiques.

Il existe plusieurs types de serveurs, chacun ayant des fonctions spécifiques :

- **Serveur statique** : Ce type de serveur se contente de servir des fichiers HTML, CSS, et JavaScript. Il ne fait aucun traitement particulier ; il livre simplement le contenu statique tel qu'il est stocké.

- **Serveur dynamique** : Il génère du contenu à la volée en fonction des requêtes, en utilisant des langages côté serveur comme PHP, Python, ou Node.js. Exemple : Express.js, Django.
- **Serveur d'applications** : Pour des applications complexes, ce serveur prend en charge la logique d'affaires, les bases de données, et sert du contenu dynamique.
- **Serveur de base de données** : Bien qu'il ne gère pas l'affichage des pages, il stocke les données qu'un serveur d'application peut utiliser.

Chaque type de serveur joue un rôle spécifique et est souvent combiné pour créer une architecture complète (par exemple, une API RESTful côté serveur combinée à un serveur de base de données).

En résumé, un serveur est nécessaire pour bien exécuter une page HTML en raison de la gestion des requêtes, de la structure des fichiers, et des optimisations de performance qui sont impossibles à réaliser en l'ouvrant simplement en local.

## CHAPITRE 3 : ELEMENTS DE TEXTE

### PARAGRAPHES

Les paragraphes sont définis avec l'élément `<p>`. Ils permettent de structurer le texte en blocs et de le rendre plus lisible. Chaque paragraphe est isolé par une ligne vide dans de nombreux navigateurs pour une séparation visuelle (ex : `<p>Ceci est un paragraphe.</p>`).

En HTML, il est important de comprendre que les moteurs de recherche et les navigateurs ignorent les espaces blancs lorsqu'ils interprètent du code HTML. C'est pourquoi chaque paragraphe doit être entouré de balises `<p>`. Si vous voulez aller à la ligne, cela signifie en fait que vous voulez commencer un nouveau paragraphe, donc vous devez fermer le précédent et en ouvrir un nouveau.

On peut utiliser `<br />` pour un retour rapide, mais il est recommandé de privilégier des paragraphes bien définis pour une meilleure lisibilité et une mise en page plus propre. Ça garantit aussi une meilleure compréhension du contenu par les navigateurs et les moteurs de recherche (ex :

```
<p>Ceci est mon premier paragraphe. Il contient une idée complète.</p>
<p>Et voici le paragraphe suivant, qui commence une nouvelle pensée.</p>
<p>Enfin, un autre paragraphe pour bien structurer le contenu.</p>
```

En organisant le texte de cette manière, chaque section sera bien séparée et lisible, autant pour les utilisateurs que pour le navigateur.

### EN-TETES

Les en-têtes vont de `<h1>` (le plus important) à `<h6>` (le moins important). Ils définissent la hiérarchie des titres dans une page, permettant aux utilisateurs et aux moteurs de recherche de comprendre la structure du contenu. Chaque niveau est utilisé pour signaler l'importance et le sous-thème d'une section de la page (ex :

```
<h1>Titre principal</h1>
<h2>Sous-titre</h2>.
```

### SAUTS

L'élément `<hr>` crée une ligne horizontale pour marquer une séparation thématique entre des sections. Elle est souvent utilisée pour indiquer un changement de sujet ou pour diviser visuellement les parties d'un document (ex :

```
<p>Première section de texte.</p>
<hr>
<p>Deuxième section de texte.</p>.
```

L'élément `<br>` en HTML, connue sous le nom de « saut de ligne », est utilisée pour insérer un retour à la ligne dans le texte. Elle est souvent utilisée pour forcer un retour à la ligne à un endroit spécifique dans le texte, sans commencer un nouveau paragraphe. Cela peut être utile dans des situations telles que les adresses, les poèmes ou les paroles de chansons, les instructions où un espacement spécifique est souhaité (ex :

```
<p>Voici une adresse :</p>
<p>123 Rue Exemple<br>Kinshasa<br>Mbok 'Elengi</p>).
```

## LISTES

Les listes en HTML sont des éléments essentiels pour structurer des informations sous forme de points ou de séquences numérotées. Elles facilitent la lecture et permettent de présenter les informations de manière concise et organisée. Il existe trois types principaux de listes en HTML : les listes non ordonnées, les listes ordonnées, et les listes de définition.

### LISTES NON ORDONNEES

Les listes non ordonnées sont utilisées pour afficher des éléments sans ordre spécifique, souvent représentés par des puces. Elles sont utiles lorsque l'ordre des éléments n'a pas d'importance. L'élément principal est `<ul>` et il doit contenir un ou plusieurs éléments de liste `<li>` (list item).

```
<ul>
  <li>Élément 1</li>
  <li>Élément 2</li>
  <li>Élément 3</li>
</ul>). Cela affichera les éléments sous forme de liste à puces :
```

- Élément 1
- Élément 2
- Élément 3

### LISTES ORDONNEES

Les listes ordonnées sont utilisées lorsque les éléments suivent un ordre ou une séquence logique. Par exemple, une liste de tâches ou des étapes d'instruction. L'élément principal est `<ol>` et il doit contenir un ou plusieurs éléments de liste `<li>` (list item). `<ol>` accepte jusqu'à trois attributs qui lui sont propres, en dehors des attributs globaux :

1. **type** : Définit le style de numérotation. Elle prend comme valeur "`1`" (chiffres arabes), "`a`" (lettres alphabétiques minuscules), "`A`" (lettres alphabétiques majuscules), "`i`" (chiffres romains minuscules) ou "`I`" (chiffres romains majuscules).
2. **start** : Indique le numéro de départ pour la numérotation. Elle prend comme valeur un nombre entier contenu entre deux guillemets (ex : "`36`").
3. **reversed** : Affiche la liste dans l'ordre décroissant. Elle ne prend aucune valeur, sa présence ou son absence indique l'essentiel.

(Ex :

```
<ol start="3" type="A">
  <li>Élément 1</li>
  <li>Élément 2</li>
  <li>Élément 3</li>
</ol>
```

). Cela affichera les éléments sous forme de liste ordonnée avec des lettres :

- C. Élément 1
- D. Élément 2
- E. Élément 3

### LISTES DE DEFINITION

Les listes de définitions sont spécifiquement conçues pour afficher des termes suivis de leurs descriptions, comme un dictionnaire où les informations sont structurées en paires (terme et définition). L'élément principal est `<dl>` et il doit contenir un ou plusieurs éléments de terme `<dt>` et un ou plusieurs éléments de description `<dd>`.

(Ex :

```
<dl>
  <dt>JavaScript</dt>
  <dt>HTML</dt>
  <dd>HyperText Markup Language</dd>
  <dd>HyperText Markup Language</dd>
  <dt>CSS</dt>
  <dd>Cascading Style Sheets</dd>
</dl>). Cela affichera chaque terme avec sa description en retrait en dessous :
```

JavaScript  
 HTML  
   Hypertext Markup Language  
   Hypertext Markup Language  
 CSS  
   Cascading Style Sheets

Un élément de terme qui n'est pas immédiatement suivi d'un ou plusieurs éléments de description est considéré comme n'en ayant pas. Il faut faire attention à ce que ce ne soit pas le cas involontairement.

## LISTES IMBRIQUEES

Les listes peuvent être imbriquées pour créer des sous-listes. Cela permet d'organiser des éléments hiérarchiquement (ex :

```
<ul>
  <li>Fruits
    <ul>
      <li>Pomme</li>
      <li>Banane</li>
    </ul>
  </li>
  <li>Légumes
    <ol>
      <li>Carotte</li>
      <li>Brocoli</li>
    </ol>
  </li>
</ul>). Cela affichera les éléments de manière hiérarchique, avec des sous-catégories :
```

- Fruits
  - Pomme
  - Banane
- Légumes
  1. Carotte
  2. Brocoli

## ELEMENTS DE TEXTE DIVERS

Il existe plusieurs éléments secondaires qui permettent de formater du texte d'une manière ou d'une autre, notamment `<blockquotes>`, `<pre>`, `<figure>`, `<figurecaption>`, etc.

Pour en savoir plus sur ces éléments, on peut aller sur [MDN Web Docs](#) ou sur [W3Schools](#).

## ELEMENTS DE TEXTE EN LIGNE

Les éléments de texte en ligne en HTML permettent de formater et de styliser des parties spécifiques de texte sans affecter la structure générale de la page. Contrairement aux éléments de bloc, les éléments en ligne ne provoquent pas de retour à la ligne, ce qui signifie qu'ils peuvent être utilisés à l'intérieur de paragraphes ou d'autres éléments de texte. Voici une liste des principaux éléments de texte en ligne en HTML :

---

## TEXTE EMPHATIQUE

L'élément `<em>` est utilisé pour donner de l'importance à un texte en le rendant emphatique. Les navigateurs le stylisent généralement en italique, mais il est aussi interprété par les lecteurs d'écran pour indiquer un contenu important (ex :

`<p>Il est <em>très</em> important de suivre les instructions.</p>`). Cela affiche : « Il est très important de suivre les instructions. »

---

## TEXTE FORT OU IMPORTANT

L'élément `<strong>` indique une forte importance pour le texte, et il est souvent affiché en gras par défaut. Les lecteurs d'écran peuvent aussi l'annoncer avec une tonalité plus prononcée, ce qui en fait un élément sémantiquement fort (ex :

`<p><strong>Attention</strong> : Ne pas utiliser sans précaution.</p>`). Cela affiche : « **Attention** : Ne pas utiliser sans précaution. »

---

## TEXTE EN ITALIQUE

L'élément `<i>` met le texte en italique sans signification particulière, contrairement à `<em>`. Il est souvent utilisé pour des titres d'ouvrages, des termes étrangers, ou pour styliser du texte (ex :

`<p>Le mot italien <i>ciao</i> signifie "salut".</p>`). Cela affiche : « Le mot italien *ciao* signifie "salut". »

---

## TEXTE EN GRAS

L'élément `<b>` met en gras un texte sans signification spécifique. Contrairement à `<strong>`, il ne donne pas de poids sémantique au texte ; il est simplement stylisé en gras (ex :

`<p>Les mots clés sont : <b>sécurité</b> et <b>efficacité</b>.</p>`). Cela affiche : « Les mots clés sont : **sécurité** et **efficacité**. »

---

## TEXTE SURLIGNE

L'élément `<mark>` est utilisé pour mettre en évidence du texte, comme si on l'avait surligné avec un marqueur. Il est utile pour attirer l'attention sur une portion de texte importante dans un contexte donné (ex :

`<p>Le point <mark>le plus important</mark> de cette discussion est l'économie.</p>`). Cela affiche : « Le point **le plus important** de cette discussion est l'économie. »

---

## TEXTE DE PETITE TAILLE

L'élément `<small>` est utilisé pour afficher du texte en plus petite taille. Il est souvent utilisé pour des notes en bas de page ou des avertissements, comme les droits d'auteur ou les clauses (ex :

`<p>Produit sous licence. <small>Conditions générales d'utilisation appliquées.</small></p>`). Cela affiche : « Produit sous licence. Conditions générales d'utilisation appliquées. »

---

## INDICES ET EXPOSANTS

L'élément `<sub>` affiche le texte en indice, comme pour les formules chimiques (ex :

`<p>Formule : H<sub>2</sub>O</p>`). Cela affiche : « Formule : H<sub>2</sub>O »

L'élément `<sup>` affiche le texte en exposant, souvent utilisé pour les puissances mathématiques ou les références (ex : `<p>Mathématiques : x<sup>2</sup>+y<sup>2</sup></p>`). Cela affiche : « Mathématiques : x<sup>2</sup>+y<sup>2</sup> »

---

## TEXTE DE CODE

L'élément `<code>` est utilisé pour afficher du code informatique ou des commandes. Le texte contenu est généralement affiché en police monospace (ex :

<p>Utilisez la commande <code>git commit -m "message"</code> pour enregistrer les changements.</p>). Cela affiche : « Utilisez la commande git commit -m "message" pour enregistrer les changements. »

## CITATIONS COURTES

L'élément <q> est utilisé pour des citations courtes, souvent intégrées dans le texte. Les navigateurs ajoutent automatiquement des guillemets autour du contenu (ex :

<p>Il dit <q>je serai là bientôt</q>.</p>). Cela affiche : « Il dit « je serai là bientôt ». »

## ABREVIATIONS

L'élément <abbr> est utilisé pour les abréviations et les acronymes. Il permet d'ajouter une explication (via l'attribut title) que les utilisateurs peuvent voir en survolant l'abréviation (ex :

<p>L'<abbr title="HyperText Markup Language">HTML</abbr> est un langage de balisage.</p>). Cela affiche : « L'HTML ». Lors du survol de ce dernier, le texte « HyperText Markup Language » apparaît.

## DATE ET HEURE

L'élément <time> est utilisé pour spécifier des dates et des heures dans un format reconnaissable par les machines, comme pour les calendriers ou les événements.

Il prend, en dehors des attributs globaux, un attribut spécifique : **datetime**. Cet attribut reçoit une valeur textuelle de format "**numAnnée-numMois-numJourTheure:minute:seconde**" et correspond à la date renseignée (ex : <p>Rendez-vous le <time datetime="2024-11-01">1er novembre 2024</time>.</p>). Cela affiche : « Rendez-vous le 1er novembre 2024. »

Les éléments de texte en ligne en HTML sont essentiels pour structurer et formater le texte de manière sémantique, tout en maintenant l'organisation de la page. Ils permettent de rendre le texte plus accessible et compréhensible pour les utilisateurs et les technologies d'assistance. Cependant, il est conseillé de réserver la mise en forme exclusivement au CSS.

## ELEMENTS GENERIQUES

Ils sont principalement utilisés pour structurer et organiser le contenu de manière flexible, et pour appliquer des styles CSS ou du JavaScript. Bien qu'ils ne communiquent pas de sens particulier aux moteurs de recherche ou aux lecteurs d'écran, ils sont essentiels pour la mise en page et le style. Il y en a deux :

### ÉLEMENT GENERIQUE DE BLOC

Le <div> (abréviation de « division ») est un conteneur de **bloc** générique. Cela signifie qu'il occupe toute la largeur disponible et crée une nouvelle ligne avant et après lui. Il est idéal pour regrouper des éléments en blocs logiques, permettant ainsi de structurer la page en sections plus cohérentes.

Les <div> sont souvent utilisés pour :

- Encadrer des sections complètes du contenu, comme des en-têtes, des pieds de page, des menus, ou des articles.
- Structurer des mises en page flexibles avec CSS en combinaison avec des systèmes de grilles ou de colonnes.
- Appliquer des styles CSS ou des scripts JavaScript à un groupe d'éléments.

(Ex :

```
<div class="header">
  <h1>Bienvenue</h1>
  <p>Ceci est le site d'exemple</p>
</div>).
```

## ÉLÉMENT GÉNÉRIQUE EN LIGNE

Le `<span>` est un conteneur **en ligne**, ce qui signifie qu'il n'affecte pas la structure du texte environnant. Il ne provoque pas de saut de ligne et peut être utilisé pour styliser ou manipuler des parties spécifiques de texte ou d'autres éléments en ligne sans altérer la mise en page globale.

Les `<span>` sont souvent utilisés pour :

- Appliquer des styles CSS spécifiques à une partie du texte, comme un mot ou une phrase dans un paragraphe.
- Manipuler du contenu en ligne de façon ciblée avec JavaScript.
- Encadrer de petites portions de texte ou d'icônes pour les styliser différemment.

(Ex : `<p>Le mot <span class="important">important</span> est en gras.</p>`).

Les éléments `<div>` et `<span>` sont souvent utilisés en association avec des classes ou des identifiants (`class` ou `id`) pour permettre des styles ou des comportements dynamiques en CSS et JavaScript.

## ELEMENTS D'ORGANISATION DU DOCUMENT

Ils permettent d'organiser nos documents sans toujours avoir à utiliser des `<div>`. Cela est très important pour le SEO. Il est donc conseillé de les utiliser autant qu'on peut lorsqu'il s'agit d'organiser nos pages. Les éléments d'organisation de document sont :

1. `<main>` : Contient le contenu principal de la page, généralement unique par page. Il exclut les éléments répétitifs comme les en-têtes et les barres de navigation.
2. `<header>` et `<footer>` : En-tête et pied de page de la page ou d'une section.
3. `<nav>` : Zone dédiée à la navigation.
4. `<article>` : Section de contenu autonome, comme un article ou un billet de blog.
5. `<section>` : Groupe de contenu thématique, utile pour organiser le contenu en plusieurs parties.

(Ex :

```
<header>En-tête du site</header>
<nav>Liens de navigation</nav>
<main>
  <article>
    <header>Titre de l'article</header> <p>Contenu de l'article.</p>
    <footer>Fin de l'article</footer>
  </article>
</main>
<footer>Pied de page du site</footer>).
```

## SEQUENCES D'ECHAPPEMENT

Les caractères d'échappement en HTML, aussi appelés « entités HTML », permettent d'afficher des caractères spéciaux qui sont normalement interprétés comme du code HTML. Ils sont utilisés pour éviter toute confusion entre le contenu et les balises HTML, et pour afficher certains symboles spéciaux.

Pour une liste complète, voir <https://mateam.net/html-escape-characters/>.

## CHAPITRE 4 : LIENS HYPERTEXTES

Les liens permettent de lier une ressource externe ou interne à une page HTML. Les liens sont utiles pour naviguer entre différentes pages web d'un site, entre différentes sections d'une même page, pour télécharger des fichiers à partir de leur URL, etc. Pour se faire, un lien s'utilise à travers un élément spécialement créé pour : **L'encre ou Anchor** (`<a>`). Voici les attributs propres à `<a>`, en dehors des attributs globaux :

1. **href** (obligatoire) : cet attribut spécifie l'URL de destination du lien. Vous pouvez utiliser des chemins relatifs (par exemple, "`./maPage.html`" pour une page située dans le même répertoire) ou des chemins absous (par exemple, "`https://exemple.com/maPage.html`"). Avec **href**, on peut :
  - Lier une page de votre propre site. On utilise pour cela des chemins relatifs (ex : `<a href="../pages/page3.html">Aller à la page 3</a>`).
  - Lier une page externe (ex : `<a href="https://google.com">Aller à google</a>`).
  - Lier à un fragment ou un point spécifique d'une page en utilisant un **id** (ex : `<a href="../pages/page3.html#conclusion">Aller vers la section conclusion de la page 3.</a>`).
2. **download** : cet attribut permet d'indiquer que le lien doit télécharger un fichier au lieu de naviguer vers une page. Vous pouvez spécifier un nom personnalisé pour le fichier téléchargé (ex : `<a href="document.pdf" download="Guide.pdf">Télécharger le guide</a>`).
3. **hreflang** : il spécifie la langue de la page liée (ex : `<a href="https://google.com" hreflang="fr">Aller à google</a>`) "fr" pour le français. Cela peut être utile pour les moteurs de recherche ou les sites multilingues.
4. **ping** : cet attribut contient une liste d'URL vers lesquelles des notifications seront envoyées lorsque l'utilisateur clique sur le lien. Cela peut être utilisé pour le suivi ou l'analyse (ex : `<a href="https://exemple.com" ping="https://analytics.com/track">Lien suivi</a>`).
5. **rel** : il spécifie la relation entre la page actuelle et la ressource liée. Par exemple :
  - "`nofollow`" : indique aux moteurs de recherche de ne pas suivre le lien.
  - "`noopener noreferrer`" : améliore la sécurité pour les liens externes ouverts avec l'attribut `target="_blank"`.
  - "`alternate`" : utilisé pour les liens vers des versions alternatives de la page (comme des flux RSS).
6. **target** : il définit où ouvrir le lien. Les valeurs courantes sont :
  - "`_self`" : ouvre le lien dans la même fenêtre ou le même onglet (valeur par défaut).
  - "`_blank`" : ouvre le lien dans une nouvelle fenêtre ou un nouvel onglet.
  - "`_parent`" : ouvre le lien dans la fenêtre ou le cadre parent.
  - "`_top`" : ouvre le lien dans une toute nouvelle fenêtre.
  - *N'importe quelle autre chaîne de caractère ne commençant pas par un « \_ »* : ouvre le lien dans un nouvel onglet et lui donne cette chaîne comme nom (ex : "`telechargement`" ).
7. **type** : cet attribut indique le type MIME de la ressource liée. (Ex : `type="application/pdf"` informe que le lien pointe vers un fichier PDF). Cela peut aider les navigateurs et les outils d'accessibilité à traiter le lien correctement. Une liste complète des valeurs valides pour cet attribut est disponible à <https://www.iana.org/assignments/media-types/media-types.xhtml>.

En dehors de l'attribut **href** qui est obligatoire, tous les autres attributs de `<a>` peuvent être utilisés en même temps ou pas, au gré de l'utilisation souhaitée (ex :

```
<a href="https://exemple.com/document.pdf"
  download="mon_document.pdf"
  hreflang="fr"
  ping="https://example.com/ping"
  rel="noopener noreferrer"
  target="_blank"
  type="application/pdf">Télécharger le document</a>
```

Ce lien, lorsqu'il est cliqué, ouvrira le document dans un nouvel onglet, téléchargera le fichier avec le nom spécifié, et enverra un ping pour le suivi.

## CHAPITRE 5 : IMAGES

L'intégration des images est essentielle dans la conception des sites web. Les images servent à enrichir l'expérience utilisateur, transmettre des messages visuels et améliorer l'esthétique des pages.

## FORMATS D'IMAGES POUR LE WEB

Les formats d'images jouent un rôle clé dans la vitesse de chargement, la qualité visuelle et la compatibilité de nos pages Web. Voici un aperçu des formats couramment utilisés :

1. **JPEG** : Utilise une compression avec perte pour des fichiers légers. Il est parfait pour les photographies et images avec beaucoup de détails. Sa compression peut cependant entraîner une dégradation visible de la qualité.
2. **PNG** : Supporte la transparence et offre une compression sans perte. Il est utilisé pour les logos, graphiques, et illustrations. Cependant, la taille des fichiers sous ce format peut être trop grande.
3. **GIF** : Supporte l'animation et la transparence. Il est utilisé pour les petites animations comme des icônes ou des memes. Il est cependant limité à 256 couleurs, inadapté pour les images complexes.
4. **SVG** : Format vectoriel permettant des images infiniment redimensionnables sans perte de qualité. Il est utilisé pour les logos, les icônes et les graphiques interactifs. Il est cependant moins adapté pour les photographies.
5. **WebP** : Format moderne offrant des tailles de fichiers réduites avec ou sans perte de qualité. C'est une alternative performante aux JPEG et PNG. Son support est inégal dans certains anciens navigateurs.
6. **AVIF** : Nouvelle génération de compression, surpassant même WebP. S'utilise sur des sites nécessitant des performances élevées et des images de qualité. Cependant, son support est encore limité.

## L'ELEMENT <IMG/>

L'élément HTML `<img/>` est la méthode de base pour insérer une image. Il est essentiel de comprendre ses attributs principaux pour un contrôle optimal :

1. **src** (obligatoire) : Définit l'URL de l'image. Celle-ci peut être absolue ou relative (ex : ``).
2. **alt** : Fournit une description textuelle de l'image. Cet attribut est crucial pour l'accessibilité, le SEO et le fallback au cas où l'image échoue à se charger (ex : ``).
3. **crossorigin** : Si l'image dont on a besoin provient d'un site externe, cet attribut permet de spécifier si le navigateur aurait besoin de fournir des informations au préalable ou pas. `"anonymous"` permet d'indiquer qu'aucune information n'est transmise (par défaut) ; `"use-credentials"` envoie des cookies et autres informations sensibles si cela est impératif pour avoir accès à l'image (ex : ``).
4. **height** (hauteur) et **width** (largeur) : Spécifient les dimensions de l'image (en pixels). Ils sont utiles pour réservé un espace avant le chargement de l'image. Elle peut cependant déformer l'image en utilisant des proportions incorrectes (ex : ``). Il est cependant peu recommandé d'utiliser ces attributs en HTML. Etant plus en rapport avec la présentation, cette tâche peut être gérée efficacement avec du CSS.
5. **srcset** : Permet de fournir plusieurs versions d'une image pour s'adapter aux résolutions (ex : ``). L'attribut `src` définit la version de l'image par défaut à charger, ici « `pardefaut.jpg` », qui sera affichée si le navigateur ne sélectionne aucune des images spécifiées dans les attributs `srcset`. Cette image par défaut assure qu'un contenu visuel est toujours disponible, même si les autres versions ne sont pas chargées pour une raison quelconque.

L'attribut `srcset` est utilisé deux fois dans ce code pour offrir des versions différentes de l'image en fonction de deux critères distincts : la largeur de l'écran et la densité de pixels. Le premier `srcset` propose plusieurs versions de l'image en fonction de la largeur d'affichage de l'écran. Par exemple, « `./petit.jpg 480w` » indique que cette image doit être chargée si l'affichage a une largeur d'environ 480 pixels, tandis que « `./grand.jpg 1200w` » est destiné aux écrans plus grands, d'environ 1200 pixels de large. Les valeurs 480w, 800w, et 1200w sont appelées « **w-descriptors** » et elles permettent au navigateur de sélectionner la version de l'image la plus adaptée à la largeur disponible, optimisant ainsi le temps de chargement et l'apparence sur différents appareils. Le navigateur calcule la largeur d'affichage et choisit l'image qui correspond le mieux, en prenant également en compte des facteurs comme le zoom ou la taille du conteneur parent.

Le second `srcset` utilise une approche différente en tenant compte de la densité de pixels de l'écran, en proposant des versions spécifiques pour des écrans à haute résolution. Par exemple, « `./moyenNB.jpg 2x` » indique que cette version de l'image est adaptée aux écrans ayant une densité de pixels deux fois supérieure à celle d'un écran standard. Ici, les valeurs comme `1.2x`, `2x`, et `3x` sont des « **x-descriptors** », qui indiquent un facteur de densité par rapport à la résolution standard (celle pour laquelle `pardefaut.jpg` est définie). Le navigateur sélectionne l'image qui correspond le mieux à la densité de l'écran, garantissant ainsi une image nette et de haute qualité même sur les écrans Retina ou autres écrans haute définition.

6. **sizes** : Permet de fournir plusieurs tailles pour une image pour s'adapter aux différentes tailles d'écran (ex :  
``). L'image par défaut est définie par l'attribut `src` avec la valeur `"./pardefaut.jpg"`, qui sera chargée si aucune version alternative n'est spécifiée ou si le navigateur ne prend pas en compte d'autres critères de sélection.

L'attribut `sizes` est essentiel pour informer le navigateur de l'espace qu'occupera l'image dans différentes conditions d'affichage. Dans ce cas, la valeur de `sizes` est `"(max-width: 600px) 100vw, 50vw"`. Cette chaîne est composée de deux parties, séparées par une virgule, et fonctionne comme une série de règles CSS que le navigateur interprète pour déterminer la taille à laquelle l'image doit être affichée.

La première règle, « `(max-width: 600px) 100vw` », signifie que si la largeur de l'affichage (ou de l'écran) est inférieure ou égale à 600 pixels, l'image occupera 100% de la largeur de l'écran (100vw). Ici, `vw` signifie « `viewport width` », donc 100vw correspond à la totalité de la largeur visible de l'écran. Cela garantit que sur des petits écrans, comme ceux des smartphones, l'image s'étendra sur toute la largeur de l'affichage, offrant une expérience visuelle optimale.

La deuxième partie, « `50vw` », est une valeur par défaut qui s'applique lorsque la condition précédente n'est pas remplie, c'est-à-dire si la largeur de l'affichage dépasse 600 pixels. Dans ce cas, l'image occupera 50% de la largeur de l'écran (50vw). Cela est utile sur des écrans plus grands, comme ceux des tablettes ou ordinateurs, où l'image peut occuper une partie seulement de l'affichage, laissant de l'espace pour d'autres contenus.

## IMAGES RESPONSIVES

L'élément `<picture>` offre une solution avancée pour les images responsives. Il permet de définir plusieurs versions d'une image selon le type de média (`media`) ou le format (`type`). Cet élément contient généralement un `<img/>` précédé de plusieurs `<source/>` qui spécifient toutes les différentes versions d'images à appliquer (ex :

`<picture>`  
`<source src="./image.avif" type="image/avif" />`  
`<source srcset="./image.webp" type="image/webp" />`  
`<source srcset="./petit.jpg 480w, ./grand.jpg 1200w" media="(max-width: 600px)" />`  
``  
`</picture>`). L'élément `<picture>` enveloppe plusieurs balises `<source/>`, chacune spécifiant une version différente de l'image, et une balise `<img/>` de repli. Cette structure optimise l'affichage des images, en tenant compte des performances et de la qualité visuelle sur différents appareils.

La première balise `<source/>` propose l'image au format AVIF, qui est l'un des formats les plus récents et efficaces en termes de compression. Le navigateur vérifie s'il supporte le type `"image/avif"` et, si c'est le cas, il charge cette version. Cela permet de réduire la taille des fichiers et d'accélérer le chargement sans sacrifier la qualité de l'image. Si AVIF n'est pas pris en charge, le navigateur passe à la deuxième balise `<source/>`, qui offre une version WebP. Le format WebP est également très performant, offrant une bonne compression tout en maintenant une qualité visuelle élevée. Cette approche garantit que les utilisateurs bénéficient de la meilleure expérience possible, en fonction de la compatibilité de leur navigateur.

La troisième balise `<source/>` s'intéresse à la taille de l'écran. Elle contient un attribut `media="(max-width: 600px)"`, qui signifie que si l'affichage a une largeur maximale de 600 pixels, le navigateur sélectionne l'image appropriée parmi celles spécifiées dans l'attribut `srcset`. Deux options sont proposées : « `petit.jpg` » pour les écrans plus petits, et « `grand.jpg` » pour des écrans un peu plus larges mais toujours dans la limite des 600 pixels. Cette flexibilité permet d'afficher des images optimisées selon la taille de l'écran, améliorant ainsi les performances et la présentation sur les petits appareils, comme les smartphones.

Enfin, la balise `<img/>` à l'intérieur de `<picture>` sert de solution de repli. Si aucune des images spécifiées dans les balises `<source/>` n'est compatible ou ne peut être chargée, l'image par défaut « `pardefaut.jpg` » sera affichée.

## DECOUVERTE DES ATTRIBUTS DE `<SOURCE/>`

En dehors des attributs de `<img/>`, l'élément `<source/>` prend deux autres attributs très importants :

1. `media` : Sert à cibler des tailles ou orientations spécifiques avec des requêtes CSS (ex : `<source srcset="./portrait.jpg" media="(orientation: portrait)" />`).
2. `type` : Spécifie le format MIME de l'image (ex : `<source srcset="./image.webp" type="image/webp" />`).

En conclusion, `media` est parfait pour des configurations basées sur l'appareil. `type` est utilisé pour fournir des formats alternatifs en fonction de la compatibilité du navigateur.

## DESSINER DES FORMES SVG

Les SVG offrent une manière puissante de créer des graphiques vectoriels dans une page web. Ils sont redimensionnables, stylisables avec du CSS, et interactifs avec JavaScript.

Il y a deux manières de dessiner des formes SVG en HTML :

1. Inclure un fichier d'extension « `.svg` » via un `<img/>` (ex : ``);
2. Insérer le contenu d'un fichier « `.svg` » dans un élément `<svg>` (ex :  
`<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 72 72" width="100" height="100">`  
 `<circle fill="#D4AB00" cx="36" cy="36" r="36"/>`  
 `<circle opacity=".7" fill="#FFF" stroke="#8A291C" cx="36.1" cy="35.9" r="31.2"/>`  
 `<circle fill="#A52C1B" cx="25" cy="20" r="4.8"/>`  
 `<circle fill="#A52C1B" cx="47" cy="20" r="4.8"/>`  
 `<circle fill="#A52C1B" cx="20" cy="32" r="4.8"/>`  
 `<circle fill="#A52C1B" cx="52" cy="32" r="4.8"/>`  
 `<circle fill="#A52C1B" cx="25" cy="44" r="4.8"/>`  
 `<circle fill="#A52C1B" cx="47" cy="44" r="4.8"/>`  
 `<circle fill="#A52C1B" cx="36" cy="27" r="4.8"/>`  
`</svg>`). Le code à l'intérieur de l'élément `<svg>` ne s'agit pas du HTML, mais du XML. Une connaissance en ce langage est alors nécessaire pour dessiner manuellement des formes SVG. Généralement, on n'écrit pas à la main tout le code décrivant une image SVG (cela est long et très compliqué). On utilise des éditeurs comme GIMP, Adobe Illustrator ou Inkscape pour dessiner des formes SVG et générer automatiquement le XML correspondant.



L'attribut `xmlns` déclare un espace de nom XML pour l'élément `<svg>`. Sa valeur est un lien, qui s'agit presque toujours de `http://www.w3.org/2000/svg`, qui définit toutes les balises XML utilisables à l'intérieur (ex : `<circle/>`, `<rect/>`, ...). L'attribut `viewBox` déclare une boîte dans laquelle on peut définir quatre valeurs relatives à `width` et `height`. Voir sur [MDN Web Docs](#) pour plus d'explications. Les attributs `width` et `height` déclarent la largeur et la hauteur de l'image dessinée.

## CHAPITRE 6 : TABLEAUX

Un tableau est une manière de structurer du contenu dans une matrice 2D, comme dans les éditeurs de texte. Dans un tableau, on peut mettre du texte, du média (photo, vidéo, ...), d'autres éléments, etc.

## STRUCTURE MINIMALE D'UN TABLEAU

En HTML, un tableau est constitué d'une suite de lignes (rows) où chaque ligne contient un nombre fini de cellules. On utilise 4 éléments de base pour créer un tableau : `<table>`, `<tr>`, `<th>`, `<td>`.

Ligne	Cellule	Cellule	Cellule
Ligne	Cellule	Cellule	Cellule
Ligne	Cellule	Cellule	Cellule

1. **<table>** : C'est la balise qui contient un tableau en HTML. En dehors des attributs globaux, il prend deux autres attributs : **border** et **sortable**, qu'il n'est pas recommandé d'utiliser car ils relèvent de l'aspect présentation, et donc, qu'il est préférable de gérer avec le CSS.
2. **<tr>** : C'est la balise qui représente une ligne d'un tableau en HTML. Elle se trouve toujours dans un **<table>** et ne prend aucun attribut en dehors des attributs globaux.
3. **<th>** : C'est la balise qui contient une *cellule d'en-tête*. Elle se trouve toujours dans un **<tr>**. Les attributs de cet élément seront vus au point suivant.
4. **<td>** : C'est la balise qui contient une *cellule de données*. Elle se trouve toujours dans un **<tr>**. Les attributs de cet élément seront vus au point suivant.

En remplaçant ces éléments dans le tableau ci-haut, on obtient :

```


| Cellule | Cellule | Cellule |
|---------|---------|---------|
| Cellule | Cellule | Cellule |
| Cellule | Cellule | Cellule |


```

En l'écrivant de manière linéaire, on obtient :

```


| Cellule | Cellule | Cellule |
|---------|---------|---------|
| Cellule | Cellule | Cellule |
| Cellule | Cellule | Cellule |


```

Cela crée un tableau générique de cette forme :

Cellule	Cellule	Cellule
Cellule	Cellule	Cellule
Cellule	Cellule	Cellule

Voilà la forme minimale d'un tableau en HTML avec des cellules d'en-tête et des cellules de données. Il peut cependant être réorganisé comme on veut, et c'est ce que nous allons voir par la suite.

## ETENDRE LES CELLULES D'UN TABLEAU

Parfois, on a besoin qu'une cellule occupe plusieurs colonnes ou plusieurs lignes dans un tableau. En HTML, cela est possible grâce aux attributs **colspan** et **rowspan** des éléments **<th>** et **<td>**.

1. **colspan** : Cet attribut permet à une cellule de s'étendre sur plusieurs colonnes (ex :

```


| Colonne 1 | Colonne 2 | Colonne 3 |
|-----------|-----------|-----------|
|-----------|-----------|-----------|


```

```

</tr>
<tr>
  <td colspan="2">Cellule étendue</td>
  <td>Cellule normale</td>
</tr>
</table>). Produit comme résultat :

```

Colonne 1	Colonne 2	Colonne 3
Cellule étendue		Cellule normale

2. **rowspan** : Cet attribut permet à une cellule de s'étendre sur plusieurs lignes (ex :

```

<table>
  <tr>
    <th>En-tête</th>
    <th>Colonne 1</th>
    <th>Colonne 2</th>
  </tr>
  <tr>
    <td rowspan="2">Cellule étendue</td>
    <td>Cellule 1</td>
    <td>Cellule 2</td>
  </tr>
  <tr>
    <td>Cellule 3</td>
    <td>Cellule 4</td>
  </tr>
</table>). Produit comme résultat :

```

En-tête	Colonne 1	Colonne 2
Cellule étendue	Cellule 1	Cellule 2
	Cellule 3	Cellule 4

## LES ELEMENTS DE GROUPAGE

Le HTML permet de regrouper les lignes et les colonnes d'un tableau à l'aide des balises **<thead>**, **<tbody>**, **<tfoot>**, et **<colgroup>** et **<col>**. Cela rend le tableau plus structuré et sémantiquement correct.

1. **<thead>** : Utilisé pour regrouper les lignes d'en-tête.
2. **<tbody>** : Utilisé pour regrouper les lignes de données du tableau.
3. **<tfoot>** : Utilisé pour regrouper les lignes de pied de tableau, généralement pour des totaux.
4. **<colgroup>** et **<col>** : Ces éléments permettent de définir des styles pour une ou plusieurs colonnes.

(Ex :

```

<table>
  <colgroup>
    <col class="col-id">
    <col class="col-nom">
    <col span="2">
  </colgroup>
  <thead>
    <tr>
      <th>ID</th>
      <th>Nom</th>
      <th>Âge</th>
      <th>Ville</th>
    </tr>
  </thead>
  <tbody>

```

Cela crée un tableau générique de cette forme (en supposant que le fond gris est le style donné à **col-id** et **col-nom**) :

ID	Nom	Âge	Ville
1	Michael Koyo	25	Kinshasa
2	Inola Koyo	30	Tokyo
3	Mahogany Koyo	28	Bruxelles

```

<tr>
  <td>1</td>
  <td>Michael Koyo</td>
  <td>25</td>
  <td>Kinshasa</td>
</tr>
<tr>
  <td>2</td>
  <td>Inola Koyo</td>
  <td>30</td>
  <td>Tokyo</td>
</tr>
<tr>
  <td>3</td>
  <td>Mahogany Koyo</td>
  <td>28</td>
  <td>Bruxelles</td>
</tr>
</tbody>
<tfoot>
  <tr>
    <td colspan="4">Total : 3 entrées</td>
  </tr>
</tfoot>
</table>). Les éléments <colgroup> et <col> définissent la structure des colonnes. Ici, les classes "col-id" et "col-nom" appliquent éventuellement un style de fond différent aux deux premières colonnes. <col span="2"> signifie que les deux colonnes restantes n'ont pas de styles spécifiques appliqués. <thead> contient l'en-tête du tableau avec les titres des colonnes (ID, Nom, Âge, Ville). <tbody> inclut les données principales du tableau. <tfoot> affiche le pied de tableau. Ici, il indique le nombre total d'entrées.

```

## CHAPITRE 7 : FORMULAIRES HTML

Un formulaire en HTML est un mécanisme permettant de recueillir des données de la part de l'utilisateur et de les envoyer à un serveur de backend. On utilise des formulaires pour créer de nombreux composants comme des commentaires, des formulaires d'inscription, etc.

### COMMENT LES FORMULAIRES FONCTIONNENT-T-ILS ?

Un formulaire web repose sur deux mécanismes essentiels qui assurent son bon fonctionnement. Le premier mécanisme concerne la création du formulaire en lui-même à l'aide du langage HTML. Ce formulaire utilise divers éléments interactifs tels que des champs de texte, des boutons, des cases à cocher et des listes déroulantes, permettant à l'utilisateur de saisir des informations.

Une fois les données saisies, le second mécanisme intervient : l'envoi des informations au serveur via une requête HTTP. Cette requête peut utiliser différentes méthodes, comme GET (pour récupérer des informations) ou POST (pour envoyer des données à traiter). Le backend, généralement développé avec des langages de programmation comme Java, Python, C# ou des scripts tels que PHP et Node.js, joue un rôle crucial à cette étape. Lorsqu'il reçoit une requête HTTP, le backend traite les informations reçues, effectue les opérations nécessaires (comme la validation des données, les calculs ou les interactions avec une base de données), puis génère une réponse qui est renvoyée au frontend.

### L'ELEMENT <FORM>

L'élément de base pour rajouter un formulaire à une page est <form>. Cet élément encapsule les éléments qui constituent un formulaire et surtout, grâce aux attributs qu'il accepte, indique aux navigateurs comment créer la requête qui va contenir les données entrées. Ses attributs, en dehors des attributs globaux, sont :

1. **action** (obligatoire) : Définit l'URL où les données du formulaire seront envoyées lorsque celui-ci sera soumis. Elle prend comme valeur une URL relative ou absolue (ex : "<https://example.com/form-handler>", "./submit-form"). Si aucune URL n'est spécifiée (""), la soumission s'effectue à l'URL de la page actuelle (utile dans les cas où le script de gestion du formulaire est écrit dans le même fichier, comme en PHP).
2. **method** (obligatoire) : Spécifie la méthode HTTP utilisée pour envoyer les données du formulaire. Les formulaires ne supportent que deux méthodes http : "POST" et "GET". Avec "GET", les données sont ajoutées à l'URL en tant que paramètres de requête. Convient pour des requêtes sans effet secondaire (comme des recherches). Avec "POST", les données sont envoyées dans le corps de la requête HTTP. Plus sûr pour envoyer des données sensibles au serveur.
3. **accept-charset** : Définit l'encodage des caractères que le serveur peut gérer pour les données du formulaire. La valeur par défaut est "UTF-8".
4. **autocomplete** : Contrôle si le navigateur peut suggérer automatiquement des valeurs pour les champs de formulaire basés sur les données enregistrées de l'utilisateur. On peut spécifier deux valeurs : "on" (par défaut) ou "off".
5. **enctype** : Définit comment les données du formulaire doivent être encodées avant d'être envoyées au serveur. Utilisé principalement avec la méthode POST. La valeur que peut prendre cet attribut peut être "application/x-www-form-urlencoded" (par défaut, indique que les données doivent être encodées sous forme de paires clé-valeur), "multipart/form-data" (Requis pour l'envoi de fichiers) ou "text/plain" (indique que les données ne seront pas encodées).
6. **name** : Attribue un nom au formulaire. Utile pour manipuler un formulaire via JavaScript ou en via l'attribut **target** lorsque plusieurs sont présents sur la même page. L'attribut **name** présent sur l'élément `<form>` a une utilité différente de celui des champs de formulaire individuels comme `<input>` ou `<textarea>`. Il n'est pas envoyé au serveur lorsqu'un formulaire est soumis. Seules les données des champs à l'intérieur du formulaire sont transmises.
7. **novalidate** : Indique au navigateur de ne pas valider les champs du formulaire avant la soumission. Elle ne prend aucune valeur, la simple présence de cet attribut désactive la validation.
8. **target** : Détermine où afficher la réponse après la soumission du formulaire. Les valeurs courantes sont :
  - "\_self" : ouvre la réponse dans la même fenêtre ou le même onglet (valeur par défaut).
  - "\_blank" : ouvre la réponse dans une nouvelle fenêtre ou un nouvel onglet.
  - "\_parent" : ouvre la réponse dans la fenêtre ou le cadre parent.
  - "\_top" : ouvre la réponse dans une toute nouvelle fenêtre.
  - *N'importe quelle autre chaîne de caractère ne commençant pas par un « \_ »* : ouvre la réponse dans un nouvel onglet et lui donne cette chaîne comme nom (ex : "réponse").

## LES VARIABLES DE FORMULAIRE

Lorsqu'un formulaire est soumis, les données des champs sont envoyées au serveur via une requête HTTP selon deux méthodes principales :

1. GET : Les données sont collées à l'URL sous forme de chaîne de requête de la forme « URL?cle=valeur&cle2=valeur2 » (ex : <http://exemple.com/script.php?username=Chris&password=1234>).
2. POST : Les données sont envoyées dans le corps de la requête, ce qui est plus sécurisé pour des informations sensibles.

Au backend, les données de ces champs sont stockées dans des variables de formulaire. Ces derniers sont constituées de paires clé-valeur où chaque clé correspond à l'attribut **name** de chaque champ du formulaire et chaque valeur correspond à la valeur saisie ou sélectionnée par l'utilisateur pour chaque champ.

En PHP par exemple, ces variables sont accessibles au backend via des « tableaux superglobaux » :

1. **`$_GET`** : Contient les variables transmises via la méthode GET.
2. **`$_POST`** : Contient les variables transmises via la méthode POST.
3. **`$_REQUEST`** : Contient les variables des deux (GET et POST), mais son utilisation est déconseillée pour des raisons de sécurité.
4. **`$_FILES`** : Contient les variables contenant des informations sur les fichiers envoyés au serveur (lorsque l'attribut **enctype="multipart/form-data"** est utilisé).

Les variables de formulaire peuvent contenir différents types de données selon les champs utilisés.

## L'ELEMENT <TEXTAREA>

Cet élément permet d'insérer une zone dans le formulaire où l'utilisateur peut rentrer du texte sur plusieurs lignes. Ses attributs, en dehors des attributs globaux, sont :

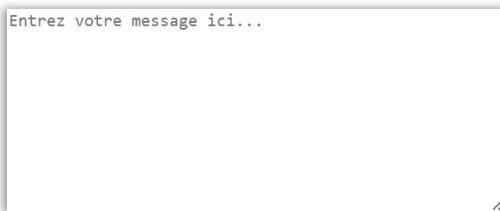
1. **name** (obligatoire) : Permet d'identifier le champ lorsqu'il est envoyé avec un formulaire. Il est utilisé pour nommer le champ de texte et le relier à des données dans un backend.
2. **autocomplete** : Contrôle la possibilité pour le navigateur de suggérer des valeurs pour l'entrée basée sur des entrées précédentes. Il peut prendre les valeurs "**on**" (par défaut) ou "**off**".
3. **autofocus** : Permet de mettre automatiquement le focus sur le champ de texte au chargement de la page. Il n'accepte pas de valeur, mais sa présence suffit pour activer le focus.
4. **cols** : Spécifie le nombre de colonnes visibles (largeur) de la zone de texte. L'unité est le nombre de caractères de texte visibles.
5. **rows** : Spécifie le nombre de lignes visibles (hauteur) de la zone de texte. C'est le nombre de lignes de texte qui seront visibles par défaut sans faire défiler.
6. **dirname** : permet de spécifier un nom pour l'attribut **dir** lors de l'envoi du formulaire. Il peut être utilisé pour envoyer l'orientation du texte dans le champ (gauche à droite ou droite à gauche).
7. **disabled** : Lorsque cet attribut est présent, il désactive la zone de texte. L'utilisateur ne pourra pas modifier le contenu ou interagir avec la zone de texte.
8. **readonly** : Rend la zone de texte en lecture seule. L'utilisateur peut voir le contenu, mais ne peut pas le modifier. Contrairement à **disabled**, un champ en lecture seule sera toujours soumis lorsqu'un formulaire est envoyé.
9. **form** : permet de lier un élément <textarea> à un formulaire particulier, même si le champ de texte est en dehors de ce formulaire. Il prend comme valeur l'**id** du formulaire auquel il appartient.
10. **inputmode** : Déclare le type d'entrée attendu, ce qui permet d'optimiser le clavier des appareils mobiles. Les valeurs possibles sont :
  - "**verbatim**" : Cette valeur indique que le champ accepte une saisie libre, sans aucune restriction particulière. Le clavier ou la méthode de saisie fournie sera générique, laissant l'utilisateur entrer n'importe quel caractère disponible.
  - "**text**" : identique à "**verbatim**".
  - "**latin**" : Spécifie que le champ attend des caractères latins de base.
  - "**latin-name**" : Cette valeur cible des noms dans les langues utilisant des caractères latins, avec des options adaptées comme la capitalisation automatique de la première lettre.
  - "**latin-prose**" : Destiné à des textes complets en caractères latins, incluant des espaces, de la ponctuation et des lettres capitales ou minuscules.
  - "**full-width-latin**" : Prévoit une saisie en caractères latins à pleine largeur, souvent utilisée dans des contextes asiatiques (comme en japonais), où les caractères pleins sont préférés pour des raisons esthétiques ou de compatibilité.
  - "**kana**" : Indique que le champ attend une saisie en kana japonais (katakana ou hiragana).
  - "**katakana**" : Cette valeur spécifie que le champ attend une saisie en katakana japonais, un système d'écriture phonétique utilisé principalement pour les mots d'origine étrangère, les noms propres ou les onomatopées.
  - "**numeric**" : Indique que le champ accepte uniquement des chiffres (0-9). Sur un appareil mobile, cela fait apparaître un clavier numérique sans symboles ou lettres.
  - "**decimal**" : Indique que le champ accepte uniquement des chiffres décimaux (avec virgules). Sur un appareil mobile, cela fait apparaître un clavier numérique sans symboles ou lettres en dehors du point ou de la virgule.
  - "**tel**" : Optimise la saisie pour un numéro de téléphone. Cela affiche un clavier numérique avec des touches supplémentaires telles que le « + », le « - », ou les parenthèses pour les formats de numéros de téléphone internationaux.
  - "**url**" : Destiné à la saisie d'une adresse URL. Cela affiche un clavier avec des touches adaptées comme « / », « . », et « @ » pour rendre l'entrée des adresses Web plus rapide et facile.
  - "**email**" : Conçu pour la saisie d'une adresse e-mail. Le clavier affiche des touches supplémentaires comme « @ » et « . », qui sont couramment utilisées dans les adresses électroniques.
  - "**search**" : Optimise la saisie pour une recherche. Cela affiche un clavier standard avec une disposition légèrement optimisée pour les recherches (souvent inclut une touche « Rechercher »).
11. **maxlength** : Définit le nombre maximal de caractères que l'utilisateur peut entrer dans la zone de texte. Si un utilisateur tente de saisir plus de caractères que la limite, le champ ignore les nouvelles entrées.

12. **placeholder** : Fournit un texte d'exemple à l'intérieur de la zone de texte, qui s'affiche lorsque celle-ci est vide. Ce texte disparaît dès que l'utilisateur commence à saisir des informations.
13. **required** : Indique que la zone de texte doit obligatoirement être remplie avant que le formulaire ne soit soumis. Si ce champ est vide, le formulaire ne sera pas envoyé.
14. **wrap** : Définit comment le texte doit être enveloppé lorsque l'utilisateur tape dans la zone de texte. L'attribut peut prendre les valeurs "**soft**" (lorsque le texte est soumis sans retour à la ligne) ou "**hard**" (lorsque le texte est soumis avec des retours à la ligne).

On utilise cet élément lorsqu'on veut recueillir du texte de la part de l'utilisateur.

(Ex :

```
<textarea
  name="message"
  autocomplete="on"
  cols="50"
  rows="10"
  form="monFormulaire1"
  inputmode="text"
  maxlength="500"
  placeholder="Entrez votre message ici..."
  wrap="soft">
</textarea>). Ce code produit la zone de texte suivante :
```



Lorsqu'on place un contenu textuel à l'intérieur de cet élément (ex : <textarea>coucou</textarea>), celui-ci est considéré comme la valeur par défaut de la variable de formulaire associée à ce <textarea>.

## L'ELEMENT <INPUT />

Cet élément permet d'ajouter plusieurs **types** de contrôles dans un formulaire. Voici les attributs, en dehors des attributs globaux, qu'on peut utiliser sur n'importe quel **type d'input** :

1. **name** (obligatoire) : Permet d'identifier l'**input** lorsqu'il est envoyé avec un formulaire. Il est utilisé pour nommer et relier l'état de l'**input** à une variable de formulaire dans un backend.
2. **type** (obligatoire) : Permet de spécifier le type d'**input** que l'on veut créer. Il peut s'agir d'un bouton, d'un sélecteur de date, d'un bouton radio, etc.
3. **value** : Permet de spécifier une valeur par défaut pour le champ. La valeur consiste généralement en du texte libre, sauf dans le cas où il s'agit d'un type lié à une date (où le format est "**numAnnée-numMois-numJourTheure:minute:seconde**").
4. **autofocus** : Indique que l'élément doit recevoir le focus automatiquement lors du chargement de la page. Il n'accepte pas de valeur, mais sa présence suffit pour activer le focus.
5. **disabled** : Lorsque cet attribut est présent, il désactive l'**input**. L'utilisateur ne pourra pas modifier le contenu ou interagir avec l'**input**.
6. **form** : Associe l'**input** à un formulaire spécifique en utilisant l'**id** du formulaire comme valeur.

### INPUT TYPE="BUTTON"

Ce type permet de créer un bouton dont l'effet doit être spécifiée via un script JavaScript. On peut utiliser des attributs de gestion d'évènement comme **onclick**, **onhover**... si la fonction Javascript qui va spécifier l'effet est écrite directement dans le fichier .html (dans un élément <script>) ou bien lui affecter un **id** qu'on peut manipuler dans un fichier .js externe qu'on pourra inclure dans notre code HTML.

(Ex :

```
<script>
  function bonjour(){
    alert('vous avez cliqué sur le bouton ' +
      document.querySelector("input").getAttribute("value"));
  }
</script>
```

<input type="button" value="Bonjour" onclick="bonjour()">. Ce code produit le bouton suivant :



Lorsqu'on clique sur le bouton, cela crée un événement `click` qui est capturée par l'attribut de gestion d'évènement comme `onclick`, celui-ci entraîne l'exécution de la fonction JavaScript `bonjour()` qui se trouve dans le même fichier HTML.

(Ex : Voici un autre exemple qui produit le même résultat mais, cette fois, utilise un `id` pour une manipulation externe :

Dans `bonjour.html` :

```
<script src="./bonjour.js"></script>
<input type="button" value="Bonjour" id="bjr">
```

Dans `bonjour.js` :

```
const bouton = document.getElementById("bjr");
bouton.addEventListener("click", ()=>(
  alert('vous avez cliqué sur le bouton ' + bouton.getAttribute("value"))
)
);
```

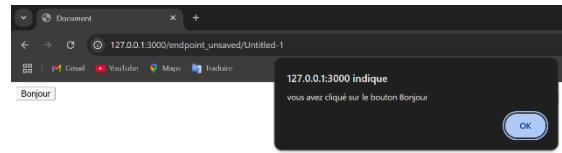


Figure 1 Résultat après clic sur le bouton.

On utilise ce type lorsqu'on veut créer un bouton dont le comportement doit être spécifié via JavaScript.

## INPUT TYPE="SUBMIT"

Cet input crée un bouton qui, une fois cliqué, permet de soumettre le formulaire auquel il appartient. Par défaut, elle envoie une requête POST avec toutes les variables de formulaire contenant les informations entrées, mais cela, ainsi que plusieurs autres paramètres peuvent être configurés grâce à des attributs spécialisées.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "`submit`" :

1. **`formaction`** : Spécifie l'URL où les données du formulaire qui contient le bouton seront envoyées.
2. **`formenctype`** : Définit comment les données du formulaire dans lequel cet `input` se trouve doivent être encodées avant d'être envoyées au serveur. Utilisé principalement avec la méthode POST. La valeur que peut prendre cet attribut peut être "`application/x-www-form-urlencoded`" (par défaut, indique que les données doivent être encodées sous forme de paires clé-valeur), "`multipart/form-data`" (Requis pour l'envoi de fichiers) ou "`text/plain`" (indique que les données ne seront pas encodées).
3. **`formmethod`** : Indique la méthode HTTP ("`GET`" ou "`POST`") utilisée lors de la soumission du formulaire qui contient cet `input`.
4. **`formnovalidate`** : Désactive la validation pour le formulaire qui contient ce bouton.
5. **`formtarget`** : Définit où afficher la réponse après la soumission du formulaire, identique à l'attribut `target` de l'élément `<form>`.

On peut utiliser ces attributs en lieu de ceux de l'élément `<form>`.

(Ex : <input type="submit" value="soumettre le formulaire">). Ce code produit le bouton suivant :

soumettre le formulaire

**Nota :** Tout formulaire doit obligatoirement avoir un bouton de type `submit`.

---

### INPUT TYPE="IMAGE"

Cet input crée une image qui peut s'utiliser comme un bouton de soumission de formulaire (`type="submit"`). Une fois cliqué, il soumet le formulaire auquel il est associé au serveur.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "`image`" :

1. `src` : Il définit l'URL de l'image.
2. `alt` : Permet de placer un texte alternatif à utiliser lorsque l'`input` ne peut pas être affichée.
3. `height` : Définit la hauteur (en pixels) de l'image.
4. `width` : Définit la largeur (en pixels) de l'image.

(Ex : `<input type="image" src="./antigone.png" alt="image de soumission">`). Ce code produit l'image suivante :



Une fois que vous cliquez sur elle, le formulaire dans lequel elle se trouve est soumis.

---

### INPUT TYPE="RESET"

Cet input crée un bouton qui, une fois cliqué, remet toutes les valeurs des entrées du formulaire auquel il appartient à leur valeur initiale (Ex : `<input type="reset" value="réinitialiser le formulaire">`).

---

### INPUT TYPE="CHECKBOX"

Cet input crée une case que l'on peut cocher ou décocher. On utilise ce genre d'input dans les formulaires lorsqu'on veut donner à l'utilisateur la possibilité de sélectionner ou de désélectionner une ou plusieurs options en même temps.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "`checkbox`" :

1. `checked` : Spécifie si le checkbox doit être présélectionné au chargement de la page. Il n'accepte pas de valeur, mais sa présence suffit.
2. `required` : Indique que le checkbox doit être coché obligatoirement avant la soumission du formulaire.

(Ex :

`<input type="checkbox" value="checkbox1coché" name="checkbox1" checked>`  
`<input type="checkbox" value="checkbox2coché" name="checkbox2">`



Si le formulaire y lié est soumis avec les checkboxes cochés, les paires `checkbox1=checkbox1coché` et `checkbox2=checkbox2coché` sont incluses dans le corps de la requête HTTP envoyée.

---

## INPUT TYPE="RADIO"

Cet input crée un bouton poussoir (similaire à celui des radio-émetteurs) qu'on peut allumer (seulement). On utilise ce genre d'input dans les formulaires lorsqu'on veut donner à l'utilisateur la possibilité de sélectionner une option parmi plusieurs dans une liste (`<ol>` ou `<ul>`), dans un élément générique (`<div>` ou `<span>`) ou dans un élément d'organisation de document (ex : `<nav>`). Contrairement au checkbox, on ne peut pas désélectionner un bouton radio qu'on a déjà sélectionné en réappuyant dessus.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "radio" :

1. **checked** : Spécifie si l'`input` doit être présélectionné au chargement de la page. Il n'accepte pas de valeur, mais sa présence suffit.
2. **required** : Indique que l'`input` doit être rempli obligatoirement avant la soumission du formulaire.

(Ex :

```
<p>Quel âge avez-vous ?</p>
<ul>
  <li><input type="radio" name="age" value="sous24" checked> en dessous de 24 </li>
  <li><input type="radio" name="age" value="25-34"> 25 à 34 ans </li>
  <li><input type="radio" name="age" value="35-44"> 35 à 44 ans </li>
  <li><input type="radio" name="age" value="over45"> plus de 45 ans </li>
</ul>). Ce code produit :
```

Quel âge avez-vous ?

- en dessous de 24
- 25 à 34 ans
- 35 à 44 ans
- plus de 45 ans

Dans cette liste, on ne peut sélectionner qu'un bouton radio à la fois. Lorsqu'on appuie sur l'un, tous les autres se désélectionnent automatiquement.

Remarquez aussi que tous les boutons de la liste partagent la variable "age", si le formulaire y lié est soumis avec le premier radio cochés par exemple, la paire `age=sous24` est incluse dans le corps de la requête HTTP envoyée.

---

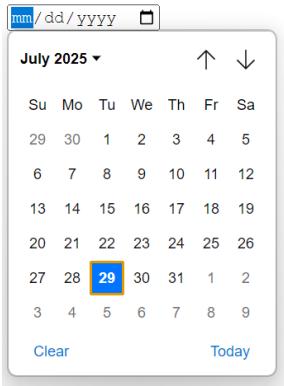
## INPUT TYPE="DATE"

Cet input crée un champ dans lequel l'utilisateur peut saisir ou sélectionner une date dans le format mois/jour/année. La valeur stockée dans la variable de formulaire est de type `string` et est écrite "année-mois-jour".

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "date" :

1. **min** : Spécifie une date minimale pour cet `input`.
2. **max** : Spécifie une date maximale pour cet `input`.
3. **step** : Indique l'intervalle de pas pour cet `input`. Il prend comme valeur n'importe quel nombre ou bien la valeur "any" (annulant son effet).
4. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "on" ou "off".
5. **list** : Indique que l'`input` à une liste de valeurs suggérées, fournies grâce à un élément `<datalist>` dont l'`id` y est utilisé comme valeur.
6. **readonly** : Empêche l'utilisateur de modifier l'`input`, mais la valeur peut être sélectionnée.

(Ex : `<input type="date" name="auj">`). Ce code produit :



Si le formulaire y lié est soumis avec la date 7/29/2025 par exemple, la paire `auj=2025-7-29` est incluse dans le corps de la requête HTTP envoyée.

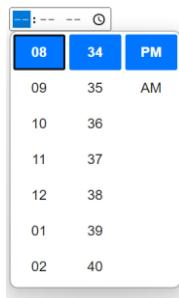
#### **INPUT TYPE="TIME"**

Cet input crée un champ dans lequel l'utilisateur peut saisir ou sélectionner une date dans le format `heure:minutes PM/AM`. La valeur stockée dans la variable de formulaire est de type `string` et est écrite "`heure:minutes`".

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "`time`" :

1. `min` : Spécifie une date minimale pour cet `input`.
2. `max` : Spécifie une date maximale pour cet `input`.
3. `step` : Indique l'intervalle de pas pour cet `input`. Il prend comme valeur n'importe quel nombre ou bien la valeur "`any`" (annulant son effet).
4. `autocomplete` : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "`on`" ou "`off`".
5. `list` : Indique que l'`input` à une liste de valeurs suggérées, fournies grâce à un élément `<datalist>` dont l'`id` y est utilisé comme valeur.
6. `readonly` : Empêche l'utilisateur de modifier l'`input`, mais la valeur peut être sélectionnée.

(Ex : `<input type="time" name="temps">`). Ce code produit :



Si le formulaire y lié est soumis avec l'heure 07:24PM par exemple, la paire `temps=19:24` est incluse dans le corps de la requête HTTP envoyée.

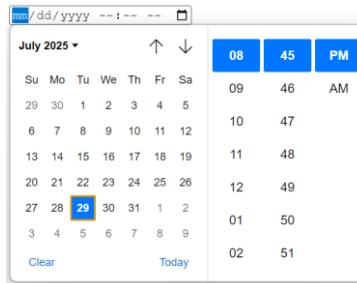
#### **INPUT TYPE="DATETIME-LOCAL"**

Cet input crée un champ dans lequel l'utilisateur peut saisir ou sélectionner une date et une heure au même moment. La valeur stockée dans la variable de formulaire est de type `string` et est écrite "`année-mois-jourTheure:minutes`".

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "`datetime-local`" :

1. **min** : Spécifie une date minimale pour cet **input**.
2. **max** : Spécifie une date maximale pour cet **input**.
3. **step** : Indique l'intervalle de pas pour cet **input**. Il prend comme valeur n'importe quel nombre ou bien la valeur "**any**" (annulant son effet).
4. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "**on**" ou "**off**".
5. **list** : Indique que l'**input** à une liste de valeurs suggérées, fournies grâce à un élément <**datalist**> dont l'**id** y est utilisé comme valeur.
6. **readonly** : Empêche l'utilisateur de modifier l'**input**, mais la valeur peut être sélectionnée.

(Ex : <**input type="datetime-local"** name="datetemps">). Ce code produit :



Si le formulaire y lié est soumis avec la date 7/29/2025 et l'heure 07:24PM par exemple, la paire **datetemps=2025-07-29T19:24** est incluse dans le corps de la requête HTTP envoyée.

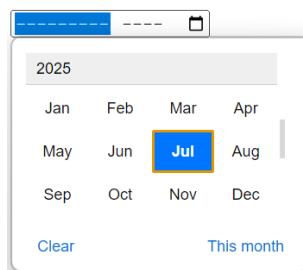
### **INPUT TYPE="MONTH"**

Cet input crée un champ dans lequel l'utilisateur peut saisir ou sélectionner un mois d'une année précise. La valeur stockée dans la variable de formulaire est de type **string** et est écrite "**année-mois**".

En plus des attributs globaux et des attributs communs à tous les types d'**input**, voici les attributs qui ne s'utilisent qu'avec les **inputs** dont le type est "**month**" :

1. **min** : Spécifie une date minimale pour cet **input**.
2. **max** : Spécifie une date maximale pour cet **input**.
3. **step** : Indique l'intervalle de pas pour cet **input**. Il prend comme valeur n'importe quel nombre ou bien la valeur "**any**" (annulant son effet).
4. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "**on**" ou "**off**".
5. **list** : Indique que l'**input** à une liste de valeurs suggérées, fournies grâce à un élément <**datalist**> dont l'**id** y est utilisé comme valeur.
6. **readonly** : Empêche l'utilisateur de modifier l'**input**, mais la valeur peut être sélectionnée.

(Ex : <**input type="month"** name="mois">). Ce code produit :



Si le formulaire y lié est soumis avec le mois Juillet 2025 par exemple, la paire **mois=2025-07** est incluse dans le corps de la requête HTTP envoyée.

---

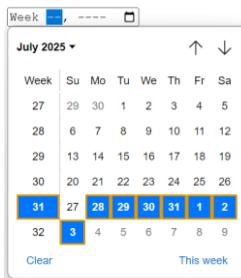
**INPUT TYPE="WEEK"**

Cet input crée un champ dans lequel l'utilisateur peut rentrer ou sélectionner une semaine d'une année précise. La valeur stockée dans la variable de formulaire est de type **string** et est écrite "[année-semaine](#)".

En plus des attributs globaux et des attributs communs à tous les types d'**input**, voici les attributs qui ne s'utilisent qu'avec les **inputs** dont le type est "**week**" :

1. **min** : Spécifie une date minimale pour cet **input**.
2. **max** : Spécifie une date maximale pour cet **input**.
3. **step** : Indique l'intervalle de pas pour cet **input**. Il prend comme valeur n'importe quel nombre ou bien la valeur "**any**" (annulant son effet).
4. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "**on**" ou "**off**".
5. **list** : Indique que l'**input** à une liste de valeurs suggérées, fournies grâce à un élément **<datalist>** dont l'**id** y est utilisé comme valeur.
6. **readonly** : Empêche l'utilisateur de modifier l'**input**, mais la valeur peut être sélectionnée.

(Ex : `<input type="week" name="sem">`). Ce code produit :



Si le formulaire y lié est soumis avec la 31<sup>ème</sup> semaine de 2025 par exemple, la paire **sem=2025-31** est incluse dans le corps de la requête HTTP envoyée.

---

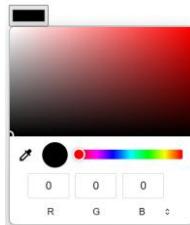
**INPUT TYPE="COLOR"**

Cet input crée un champ dans lequel l'utilisateur peut sélectionner une couleur précise ou rentrer le code RGB, HSL ou hexadécimal correspondant.

En plus des attributs globaux et des attributs communs à tous les types d'**input**, voici les attributs qui ne s'utilisent qu'avec les **inputs** dont le type est "**color**" :

1. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "**on**" ou "**off**".
2. **list** : Indique que l'**input** à une liste de valeurs suggérées, fournies grâce à un élément **<datalist>** dont l'**id** y est utilisé comme valeur.

(Ex : `<input type="color" name="col">`). Ce code produit :



Si le formulaire y lié est soumis avec la couleur de code RGB 14 144 255 par exemple, la paire **col=%23e90f** est incluse dans le corps de la requête HTTP envoyée (%23 est le caractère d'échappement du caractère #).

---

**INPUT TYPE="RANGE"**

Cet input permet à l'utilisateur de sélectionner, à l'aide d'un slider, une valeur numérique dans un intervalle donné.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "range" :

1. `min` (obligatoire) : Spécifie une valeur minimale pour cet `input`.
2. `max` (obligatoire) : Spécifie une valeur maximale pour cet `input`.
3. `step` (obligatoire) : Indique l'intervalle de pas pour cet `input`. Il prend comme valeur n'importe quel nombre ou bien la valeur "any" (annulant son effet).
4. `autocomplete` : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "on" ou "off".
5. `list` : Indique que l'`input` à une liste de valeurs suggérées, fournies grâce à un élément `<datalist>` dont l'`id` y est utilisé comme valeur.

(Ex : `<input type="range" name="abc" min="0" max="100" step="1">`). Ce code produit :



Il est souvent nécessaire d'utiliser du JavaScript et un `<label>` pour visualiser la valeur qu'on est entrain de choisir lorsqu'on manipule cet input (ex :

```
<label for="volume">Volume : <span id="valeurVolume">50</span>%</label><br>
<input type="range" id="volume" name="volume" min="0" max="100" value="50">

<script>
  const range = document.getElementById('volume');
  const affichage = document.getElementById('valeurVolume');

  range.addEventListener('input', () => {
    affichage.textContent = range.value;
  });
</script>). Ce code produit :
```



Si le formulaire y lié est soumis avec le pourcentage à 50% par exemple, la paire `volume=50` est incluse dans le corps de la requête HTTP envoyée.

---

**INPUT TYPE="FILE"**

Cet input permet à l'utilisateur d'uploader un fichier dans un formulaire. Il affiche un bouton qui ouvre une fenêtre de dialogue à travers lequel l'utilisateur sélectionne un fichier. Une fois fait, le nom de ce fichier ainsi que son format est affiché à coté de ce bouton.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "file" :

1. `accept` : Indique les types de fichiers acceptés par l'`input`.
2. `multiple` : Permet de sélectionner plusieurs fichiers ou options.

(Ex : `<input type="file" name="fichier">`). Ce code produit :



---

**INPUT TYPE="TEXT"**

Cet input permet d'insérer une zone dans le formulaire où l'utilisateur peut rentrer du texte libre. Il est comparable à l'élément <textarea>.

En plus des attributs globaux et des attributs communs à tous les types d'input, voici les attributs qui ne s'utilisent qu'avec les inputs dont le type est "text" :

1. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "on" ou "off".
2. **dirname** : Permet de spécifier un nom pour l'attribut dir lors de l'envoi du formulaire. Il peut être utilisé pour envoyer l'orientation du texte dans le champ (gauche à droite ou droite à gauche).
3. **inputmode** : Déclare le type d'entrée attendu, ce qui permet d'optimiser le clavier des appareils mobiles. Les valeurs possibles sont les même que celui de <textarea>.
4. **list** : Indique que l'input à une liste de valeurs suggérées, fournies grâce à un élément <datalist> dont l'id y est utilisé comme valeur.
5. **maxlength** : Limite le nombre maximum de caractères autorisés.
6. **pattern** : Indique une expression régulière que la valeur doit respecter.
7. **readonly** : Empêche l'utilisateur de modifier l'input, mais la valeur peut être sélectionnée.
8. **size** : Définit la largeur visible (en caractères) de l'input.
9. **placeholder** : Fournit un texte d'exemple à l'intérieur de l'input, qui s'affiche pour aider l'utilisateur à comprendre en quoi le champ consiste.

(Ex : <input type="text" name="txt" value="Entrez votre texte ici">). Ce code produit :

Si le formulaire y lié est soumis avec le texte « bonjour » par exemple, la paire **txt=bonjour** est incluse dans le corps de la requête HTTP envoyée.

---

**INPUT TYPE="EMAIL"**

Cet input permet d'insérer une zone dans le formulaire où l'utilisateur peut rentrer une adresse Mail. La différence avec l'input dont le type est "text" se trouve au niveau de l'accessibilité. Sur les écrans tactiles, cet input appelle un clavier adapté à la saisie d'adresses mail.

En plus des attributs globaux et des attributs communs à tous les types d'input, voici les attributs qui ne s'utilisent qu'avec les inputs dont le type est "file" :

1. **multiple** : Permet de sélectionner plusieurs fichiers ou options.
2. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "on" ou "off".
3. **list** : Indique que l'input à une liste de valeurs suggérées, fournies grâce à un élément <datalist> dont l'id y est utilisé comme valeur.
4. **maxlength** : Limite le nombre maximum de caractères autorisés.
5. **pattern** : Indique une expression régulière que la valeur doit respecter.
6. **readonly** : Empêche l'utilisateur de modifier l'input, mais la valeur peut être sélectionnée.
7. **size** : Définit la largeur visible (en caractères) de l'input.
8. **placeholder** : Fournit un texte d'exemple à l'intérieur de l'input, qui s'affiche pour aider l'utilisateur à comprendre en quoi le champ consiste.

(Ex : <input type="email" name="ml">). Ce code produit :

Le clavier virtuel qu'il invoque se présente comme suit (notez la présence du bouton « .com », « @ », etc.) :



Si le formulaire y lié est soumis avec l'adresse « `bonjour@chris.com` » par exemple, la paire `ml=bonjour@chris.com` est incluse dans le corps de la requête HTTP envoyée.

### **INPUT TYPE="TEL"**

Cet input permet d'insérer une zone dans le formulaire où l'utilisateur peut rentrer un numéro de téléphone. La différence avec l'`input` dont le type est "`text`" se trouve au niveau de l'accessibilité. Sur les écrans tactiles, cet input appelle un clavier numérique, adapté à la saisie de numéros de téléphone.

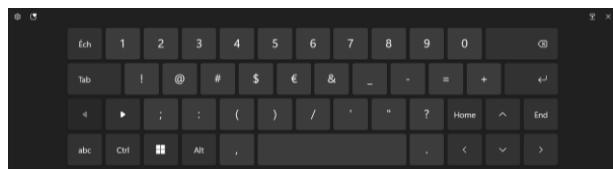
En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "`tel`" :

1. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "`on`" ou "`off`".
2. **list** : Indique que l'`input` à une liste de valeurs suggérées, fournies grâce à un élément `<datalist>` dont l'`id` y est utilisé comme valeur.
3. **maxlength** : Limite le nombre maximum de caractères autorisés.
4. **pattern** : Indique une expression régulière que la valeur doit respecter.
5. **readonly** : Empêche l'utilisateur de modifier l'`input`, mais la valeur peut être sélectionnée.
6. **size** : Définit la largeur visible (en caractères) de l'`input`.
7. **placeholder** : Fournit un texte d'exemple à l'intérieur de l'`input`, qui s'affiche pour aider l'utilisateur à comprendre en quoi le champ consiste.

(Ex : `<input type="tel" name="phone">`). Ce code produit :



Le clavier virtuel qu'il invoque se présente comme suit :



Si le formulaire y lié est soumis avec le numéro « `0899364343` » par exemple, la paire `phone=0899364343` est incluse dans le corps de la requête HTTP envoyée.

### **INPUT TYPE="SEARCH"**

Cet input permet d'insérer une zone dans le formulaire où l'utilisateur peut rentrer un texte à rechercher. La différence avec l'`input` dont le type est "`text`" se trouve au niveau de l'accessibilité. Sur les écrans tactiles, cet input appelle un clavier adapté à la recherche.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "`search`" :

1. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "`on`" ou "`off`".
2. **dirname** : permet de spécifier un nom pour l'attribut `dir` lors de l'envoi du formulaire. Il peut être utilisé pour envoyer l'orientation du texte dans le champ (gauche à droite ou droite à gauche).

3. **inputmode** : Déclare le type d'entrée attendu, ce qui permet d'optimiser le clavier des appareils mobiles. Les valeurs possibles sont les même que celui de `<textarea>`.
4. **list** : Indique que l'`input` à une liste de valeurs suggérés, fournies grâce à un élément `<datalist>` dont l'`id` y est utilisé comme valeur.
5. **maxlength** : Limite le nombre maximum de caractères autorisés.
6. **pattern** : Indique une expression régulière que la valeur doit respecter.
7. **readonly** : Empêche l'utilisateur de modifier l'`input`, mais la valeur peut être sélectionnée.
8. **size** : Définit la largeur visible (en caractères) de l'`input`.
9. **placeholder** : Fournit un texte d'exemple à l'intérieur de l'`input`, qui s'affiche pour aider l'utilisateur à comprendre en quoi le champ consiste.

(Ex : `<input type="search" name="srch">`). Ce code produit :



Le clavier virtuel qu'il invoque se présente comme suit (notez la présence du bouton « rechercher ») :



Si le formulaire y lié est soumis avec la recherche « pain » par exemple, la paire `srch=pain` est incluse dans le corps de la requête HTTP envoyée.

### **INPUT TYPE="URL"**

Cet input permet d'insérer une zone dans le formulaire où l'utilisateur peut rentrer une adresse web (lien). Sur les écrans tactiles, cet input appelle un clavier adapté à la saisie d'adresses web.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "url" :

1. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "on" ou "off".
2. **list** : Indique que l'`input` à une liste de valeurs suggérés, fournies grâce à un élément `<datalist>` dont l'`id` y est utilisé comme valeur.
3. **maxlength** : Limite le nombre maximum de caractères autorisés.
4. **pattern** : Indique une expression régulière que la valeur doit respecter.
5. **readonly** : Empêche l'utilisateur de modifier l'`input`, mais la valeur peut être sélectionnée.
6. **size** : Définit la largeur visible (en caractères) de l'`input`.
7. **placeholder** : Fournit un texte d'exemple à l'intérieur de l'`input`, qui s'affiche pour aider l'utilisateur à comprendre en quoi le champ consiste.

(Ex : `<input type="url" name="link">`). Ce code produit :



Le clavier virtuel qu'il invoque se présente comme suit (notez la présence du bouton « .com ») :



Si le formulaire y lié est soumis avec l'adresse « you.com » par exemple, la paire `link=you.com` est incluse dans le corps de la requête HTTP envoyée.

### **INPUT TYPE="PASSWORD"**

Cet input permet d'insérer une zone dans le formulaire où l'utilisateur peut rentrer un mot de passe. Visuellement, tout ce que l'utilisateur entre dans le champ est remplacé par des « ● », rendant impossible la lecture ou la copie du texte qu'on saisit. Sur les écrans tactiles, cet input appelle un clavier adapté à la saisie d'adresses web.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "password" :

1. **autocomplete** : Permet de contrôler si le navigateur doit compléter automatiquement les données. Les valeurs possibles sont "on" ou "off".
2. **pattern** : Indique une expression régulière que la valeur doit respecter.
3. **readonly** : Empêche l'utilisateur de modifier l'`input`, mais la valeur peut être sélectionnée.
4. **size** : Définit la largeur visible (en caractères) de l'`input`.
5. **placeholder** : Fournit un texte d'exemple à l'intérieur de l'`input`, qui s'affiche pour aider l'utilisateur à comprendre en quoi le champ consiste.

(Ex : `<input type="password" name="mdp">`). Ce code produit :



Le clavier virtuel qu'il invoque se présente comme suit :



Si le formulaire y lié est soumis avec le mot de passe « mamae » par exemple, la paire `mdp=mamae` est incluse dans le corps de la requête HTTP envoyée.

Il sied de préciser que cet input ne sécurise le mot de passe que visuellement, ce dernier reste visible dans la requête http émise, le rendant vulnérable aux attaques du type man-in-the-middle. D'où le besoin de mesures de sécurité supplémentaires (utilisation de HTTPS, hachage avec SHA-256 avant envoi) qu'on ne va pas exposer dans ce document.

### **INPUT TYPE="NUMBER"**

Cet input permet d'insérer une zone dans le formulaire où l'utilisateur peut rentrer un nombre. Sur les écrans tactiles, cet input appelle un clavier numérique, avec de capacités d'incrémentation, etc.

En plus des attributs globaux et des attributs communs à tous les types d'`input`, voici les attributs qui ne s'utilisent qu'avec les `inputs` dont le type est "date" :

1. **min** : Spécifie une valeur minimale pour cet `input`.
2. **max** : Spécifie une valeur maximale pour cet `input`.
3. **step** : Indique l'intervalle de pas pour cet `input`. Il prend comme valeur n'importe quel nombre ou bien la valeur "any" (annulant son effet).
4. **list** : Indique que l'`input` à une liste de valeurs suggérées, fournies grâce à un élément `<datalist>` dont l'`id` y est utilisé comme valeur.
5. **readonly** : Empêche l'utilisateur de modifier l'`input`, mais la valeur peut être sélectionnée.

(Ex : `<input type="number" name="nombre">`). Ce code produit :



Le clavier virtuel qu'il invoque se présente comme suit :



Si le formulaire y lié est soumis avec le nombre « 10 » par exemple, la paire `nombre=10` est incluse dans le corps de la requête HTTP envoyée.

### **INPUT TYPE="HIDDEN"**

Ce champ est utile pour recueillir des informations de la part de l'utilisateur sans que ce dernier puisse lui-même les entrer. Il ne s'affiche pas visuellement sur la page, mais fait partie de la requête envoyée lors de la soumission du formulaire dans lequel il appartient.

(Ex : `<input type="hidden" name="champCache">`).

### **L'ELEMENT <DATALIST>**

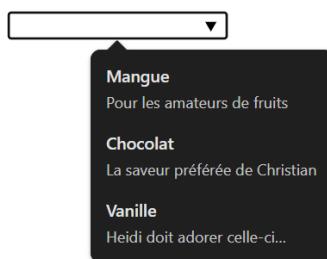
Cet élément enrichit l'élément `<input>` en lui octroyant une liste de suggestions prédéfinies que l'utilisateur peut sélectionner lorsqu'il saisit dans ce champ, sans lui obliger de choisir pour autant dans cette liste. On l'utilise souvent dans les formulaires pour la suggestion lors de la saisie de noms de pays, villes, langues, produits, ou dans des champs de recherche avec auto-complétion simple. Il n'a pas d'attributs particuliers en dehors de ceux globaux, et l'attribut `id` est obligatoire.

La mise en œuvre d'un `<datalist>` demande 2 éléments :

- `<input>` : Il s'agit de l'input auquel la liste de suggestion est reliée. Elle doit obligatoirement avoir un attribut `list` dont la valeur correspond à celle de l'attribut `id` dans le `<datalist>`.
- `<option/>` : Il s'agit de l'élément qui représente chaque suggestion dans la liste. Voici les attributs qu'il supporte en dehors des attributs globaux :
  - `label` : Permet de mettre un texte à afficher comme description de la suggestion.
  - `value` : Permet de spécifier une valeur qui sera affectée à l'`input` si l'option dans laquelle il se trouve est sélectionnée.

(Ex :

```
<input type="text" name="glace" list="saveurs">
<datalist id="saveurs">
  <option value="Mangue" label="Pour les amateurs de fruits" />
  <option value="Chocolat" label="La saveur préférée de Christian" />
  <option value="Vanille" label="Heidi doit adorer celle-ci..." />
</datalist>). Ce code produit :
```



Nota : Malgré ces trois suggestions, l'utilisateur peut quand même écrire ce qu'il veut librement.

## L'ELEMENT <SELECT>

Contrairement à `<datalist>` qui ne fait que suggérer à l'utilisateur, en lui laissant la liberté de saisir ce qu'il veut, l'élément `<select>` permet de donner à l'utilisateur le choix entre plusieurs options prédéfinies sans lui donner le droit de saisir autre chose.

Il prend en plus des attributs globaux, deux attributs qui lui sont propres :

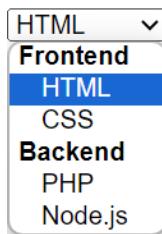
- **multiple** : Permet de sélectionner plusieurs options à la fois. Dans ce cas, la touche `Ctrl` doit être utilisée pour sélectionner plusieurs éléments.
- **size** : Définit combien d'options doivent être visibles en même temps. Si size est supérieur à 1, la liste s'affiche en mode étendu (comme une zone de sélection verticale).

La mise en œuvre d'un `<select>` demande les 2 éléments suivants :

- **<option>** : Il s'agit de l'élément qui représente chaque choix dans la liste, contrairement à l'élément option utilisable avec `<datalist>`, ce dernier est un élément à deux balises. Voici les attributs qu'il supporte en dehors des attributs globaux :
  - **label** : Permet de mettre un texte à afficher en remplacement au contenu de l'élément.
  - **value** (Obligatoire) : Permet de spécifier une valeur envoyée si cette option est sélectionnée.
  - **selected** : Permet de spécifier une option comme sélectionnée par défaut lors du chargement de la page. Il ne prend aucune valeur, sa présence suffit.
- **<optgroup>** : Il permet de regrouper plusieurs `<option>` par thème. En dehors des attributs globaux, il ne prend qu'un seul attribut, obligatoirement :
  - **label** (Obligatoire) : Permet de spécifier le nom du groupe.

(Ex :

```
<select name="cours" multiple>
  <optgroup label="Frontend">
    <option value="html">HTML</option>
    <option value="css">CSS</option>
  </optgroup>
  <optgroup label="Backend">
    <option value="php">PHP</option>
    <option value="node">Node.js</option>
  </optgroup>
</select>). Ce code produit :
```



Si le formulaire y lié est soumis avec les options « HTML » et « PHP » choisies par exemple, les paires `cours=html` et `cours/php` sont incluses dans le corps de la requête HTTP envoyée.

## L'ELEMENT <BUTTON>

Cet élément est utilisé pour créer un bouton cliquable. Lorsqu'utilisé dans un formulaire HTML, contrairement à `<input type="submit">` ou `<input type="button">`, `<button>` peut contenir du texte, des images, ou même d'autres éléments HTML (comme des icônes). Il est plus flexible et s'utilise généralement lorsque l'on souhaite un bouton stylisé ou enrichi de contenu.

Il prend, en dehors des attributs globaux, les attributs suivants :

1. **type** : Spécifie le rôle du bouton. Il peut prendre l'une des trois valeurs suivantes :
  - **"submit"** (par défaut) : Envoie le formulaire dans lequel il est contenu.
  - **"reset"** : Réinitialise les champs du formulaire dans lequel il est contenu.
  - **"button"** : Ne fait rien automatiquement ; il doit être utilisé avec JavaScript pour exécuter une action personnalisée.
2. **disabled** : Désactive le bouton. L'utilisateur ne pourra pas cliquer dessus tant qu'il est présent.
3. **autofocus** : Donne automatiquement le focus au bouton dès que la page est chargée.
4. **name** (obligatoire) : Permet d'identifier le **button** lorsqu'il est envoyé avec un formulaire. Il est utilisé pour nommer et relier l'état du **button** à une variable de formulaire dans un backend.
5. **value** : Permet de spécifier une valeur par défaut pour le bouton si cliqué. Lorsque le type du bouton est **"button"**, cette valeur est envoyée dans la requête HTTP.
6. **form** : Associe le **button** à un formulaire spécifique en utilisant l'**id** du formulaire comme valeur.
7. **formaction** : Spécifie l'URL de l'application qui va s'occuper du traitement de requête envoyée avec les données du bouton. Il est similaire à **action** de l'élément **<form>**.
8. **formenctype** : Spécifie comment les données du formulaire seront encodées. Il est similaire à **enctype** de l'élément **<form>**.
9. **formmethod** : Spécifie la méthode HTTP utilisée pour envoyer les données du formulaire. Il est similaire à **method** de l'élément **<form>**.
10. **formnovalidate** : Indique au navigateur de ne pas valider les champs du formulaire avant la soumission. Il est similaire à **novalidate** de l'élément **<form>**.
11. **formtarget** : Détermine où afficher la réponse après la soumission du formulaire. Il est similaire à **target** de l'élément **<form>**.

(Ex :

```
<button type="button">Bonjour</button>
<button type="submit" disabled>Envoyer</button>). Ce code produit :
```

## L'ELEMENT <OUTPUT>

L'élément **<output>** est utilisé pour afficher le résultat d'un calcul ou d'une action, généralement déclenchée via JavaScript. Il ne permet pas à l'utilisateur de saisir une donnée, mais uniquement d'en voir le résultat. Il est surtout utilisé dans les cas où on veut afficher dynamiquement un calcul basé sur des champs d'un formulaire.

Il peut prendre trois attributs spécifiques en plus des attributs globaux :

1. **for** : Fait référence à l'**id** des champs du formulaire concernés par ce calcul. Il peut contenir une ou plusieurs valeurs séparées par un espace.
2. **name** : Spécifie le nom de la variable à utiliser si la valeur du **<output>** doit être envoyée avec le formulaire.
3. **form** : Permet de rattacher le **<output>** à un formulaire spécifique, même s'il n'en fait pas partie directement.

(Ex :

```
<form oninput="calcul.value = parseInt(a.value) + parseInt(b.value)">
  <input type="number" name="a" id="a" /> +
  <input type="number" name="b" id="b" /> =
  <output name="calcul" for="a b"></output>
</form>). Ce code produit :
```

## L'ELEMENT <PROGRESS>

Il est utilisé pour afficher visuellement l'avancement d'un processus en cours, comme un téléchargement, une sauvegarde ou une opération de calcul. Il se présente généralement sous la forme d'une barre de progression. Il peut être mis à jour dynamiquement avec JavaScript.

Il prend deux attributs principaux (en plus des attributs globaux) :

- **value** : Spécifie la valeur actuelle de l'avancement.
- **max** : Spécifie la valeur maximale que peut atteindre la progression. S'il n'est pas précisé, la valeur maximale par défaut est 1.

(Ex : <progress value="40" max="100">40%</progress>). Cela affiche une barre de progression à 40 % :



## L'ELEMENT <METER>

Cet élément permet d'afficher une mesure connue à l'intérieur d'une plage définie. Il est souvent utilisé pour indiquer un niveau, une performance, ou une note (par exemple : batterie, score, etc.). Contrairement à <progress>, il ne sert pas à indiquer une progression dans le temps, mais plutôt à afficher une valeur dans une plage fixe.

Il accepte plusieurs attributs spécifiques :

- **value** : La valeur actuelle à afficher.
- **min** : La valeur minimale possible (par défaut = 0).
- **max** : La valeur maximale possible (obligatoire si min ≠ 0).
- **low** : Valeur en dessous de laquelle la mesure est considérée comme basse.
- **high** : Valeur au-dessus de laquelle la mesure est considérée comme élevée.
- **optimum** : Valeur considérée comme idéale.

(Ex : <meter value="65" min="0" max="100" low="30" high="80" optimum="70">65%</meter>). Cela affiche une jauge à 65 % avec une coloration automatique basée sur les plages spécifiées :



## LES ELEMENTS D'ACCESSIBILITE POUR LES FORMULAIRES

### L'ELEMENT <LABEL>

Cet élément est utilisé pour associer un texte descriptif à un élément de formulaire. Cela améliore l'accessibilité, car les lecteurs d'écran peuvent mieux identifier les champs, et l'expérience utilisateur, car cliquer sur l'étiquette sélectionne automatiquement l'élément associé. Il peut être utilisé de deux manières :

1. En enveloppant directement le champ du formulaire (ex :  

```
<label>Nom :<br/><input type="text" name="nom" /></label>).
```
2. En pointant vers l'élément cible via l'attribut **for** (ex :  

```
<label for="nom">Nom :</label><br/><input type="text" id="nom" name="nom" />).
```

Dans ce cas, la valeur de **for** doit correspondre exactement à la valeur de l'attribut **id** du champ ciblé.

Dans les deux cas, cela produit :

Nom :

### L'ELEMENT <FIELDSET>

Cet élément est utilisé pour regrouper plusieurs éléments d'un formulaire qui sont liés entre eux (par exemple : informations personnelles, préférences, paramètres, etc.). Il encadre visuellement et logiquement un ensemble de champs avec un contour.

Il est souvent accompagné de l'élément <legend>, qui sert à donner un titre ou une légende au groupe.

(Ex :

```
<fieldset>
  <legend>Informations personnelles</legend>

  <label for="nom">Nom :</label>
  <input type="text" id="nom" name="nom" />

  <label for="age">Âge :</label>
  <input type="number" id="age" name="age" />
</fieldset>). Ce code produit :
```

The screenshot shows a form element with a legend. The legend text is "Informations personnelles". Below it are two input fields: one for "Nom" and one for "Âge". Both fields have placeholder text ("Nom :" and "Âge :") and are contained within a single form group.

L'élément <fieldset> peut aussi être désactivé à l'aide de l'attribut **disabled**, ce qui désactivera automatiquement tous les champs qu'il contient.

## CHAPITRE 8 : AUTRES ELEMENTS MULTIMEDIA

Ce chapitre se concentre sur comment insérer d'autres types de médias, en dehors des images, dans une page HTML.

### L'ELEMENT <IFRAME>

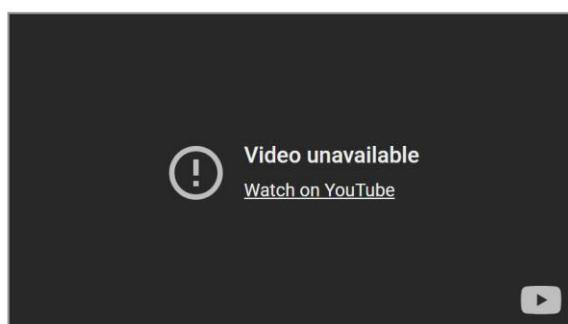
L'élément <iframe> (inline frame) permet d'insérer dans une page web une autre page web externe ou interne. Cela permet d'afficher un autre site, une carte, une vidéo, ou un document PDF directement dans la page actuelle.

Il prend plusieurs attributs, notamment :

1. **src** (obligatoire) : Spécifie l'URL de la page ou du contenu à intégrer.
2. **width** et **height** : Définissent respectivement la largeur et la hauteur du cadre.
3. **title** : Fournit un titre accessible pour les technologies d'assistance.
4. **allowfullscreen** : Permet à l'élément d'être affiché en plein écran.
5. **loading** : Peut être "**lazy**" pour différer le chargement jusqu'à ce que l'élément soit visible.

(Ex :

```
<iframe
  src="https://www.youtube.com/embed/dQw4w9WgXcQ"
  width="560"
  height="315"
  title="Vidéo YouTube"
  allowfullscreen>
</iframe>). Ce code intègre une vidéo YouTube dans la page :
```



## L'ELEMENT <CANVAS>

Cet élément permet de dessiner dynamiquement des formes, des graphiques, des animations ou des jeux 2D/3D directement dans la page HTML à l'aide de JavaScript. Il agit comme une surface de dessin vide qu'on manipule par script. Il est vide par défaut et ne contient aucun contenu visible sans script. En dehors des attributs globaux, il prend les attributs **width** et **height** qui définissent la taille de la zone de dessin.

Du contenu alternatif peut être placé entre les balises (affiché si le navigateur ne supporte pas <canvas>).

(Ex :

```
<canvas id="monCanvas" width="300" height="150">
    Votre navigateur ne supporte pas l'élément canvas.
</canvas>
```

```
<script>
const canvas = document.getElementById("monCanvas");
const contexte = canvas.getContext("2d");
contexte.fillStyle = "blue";
contexte.fillRect(50, 50, 100, 50);
</script>). Ce code produit :
```



## L'ELEMENT <OBJECT>

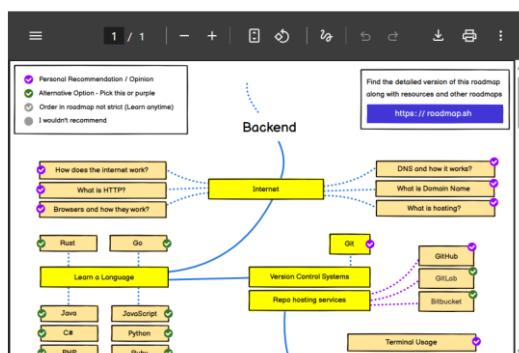
Il permet d'insérer dans une page HTML un contenu externe, comme un fichier PDF, une animation SVG, une vidéo, une page web ou tout autre type de ressource que le navigateur peut interpréter. Il offre une alternative plus générale que <iframe>, car il peut contenir tout type de média.

Il prend, en plus des attributs globaux, les attributs suivants :

1. **data** (obligatoire) : Spécifie le chemin vers la ressource externe à intégrer.
2. **type** : Définit le type MIME du contenu (ex : "application/pdf", "image/svg+xml", etc.).
3. **width** : Largeur d'affichage du contenu.
4. **height** : Hauteur d'affichage du contenu.

(Ex :

```
<object data="backend.pdf" type="application/pdf" width="600" height="400">
    Ce fichier ne peut pas être affiché.
</object>). Ce code produit :
```



## L'ELEMENT <AUDIO>

L'élément `<audio>` permet d'intégrer un fichier audio dans une page web. Il est souvent utilisé pour jouer de la musique, des messages vocaux ou des sons.

Il accepte, en dehors des attributs globaux, les attributs suivants :

1. **controls** : Affiche les contrôles du lecteur (lecture, pause, volume).
2. **autoplay** : Démarre automatiquement la lecture à l'ouverture de la page.
3. **loop** : Fait rejouer l'audio automatiquement dès qu'il se termine.
4. **muted** : Lance la lecture en mode silencieux.
5. **mediagroup** : Permet de synchroniser plusieurs éléments multimédias (ex : plusieurs audios ou vidéos) afin qu'ils soient contrôlés en groupe. Tous les éléments partageant la même valeur réagiront ensemble (pause, lecture, etc.).
6. **crossorigin** : Définit comment le navigateur gère les requêtes audios provenant d'un autre domaine. Cet attribut est utilisé lorsqu'on souhaite manipuler l'audio via JavaScript et que le fichier est hébergé sur un autre domaine. Cet attribut prend les valeurs :
  - `"anonymous"` : Aucune information d'authentification n'est envoyée.
  - `"use-credentials"` : Envoie les cookies et identifiants avec la requête.
7. **preload** : Contrôle si l'audio doit être préchargé ou non. Cet attribut prend les valeurs :
  - `"auto"` : Le navigateur est libre de décider s'il doit précharger le fichier ou non (comportement par défaut).
  - `"metadata"` : Seul le titre, la durée et les infos du fichier sont chargés, mais pas le contenu audio.
  - `"none"` : Aucune donnée n'est préchargée. Le fichier sera chargé uniquement quand l'utilisateur interagit avec.
8. **src** : Spécifie directement l'URL du fichier audio si aucun élément `<source>` n'est utilisé.

Il existe deux manières de charger un fichier audio en HTML :

1. La méthode **mono-source** : Cette méthode consiste à ne fournir qu'une seule version du fichier audio à ajouter, en spécifiant son URL dans l'attribut **src** (ex :  
`<audio src="mamusique.mp3" controls>`  
*Ce fichier ne peut pas être affiché.*  
`</audio>`). Cette méthode a un inconvénient majeur : si le format du fichier audio renseigné n'est pas supporté par le navigateur, ce dernier ne pourra pas trouver une autre alternative pour le lire.
2. La méthode **multi-source** : Cette méthode consiste à fournir plusieurs formats d'un même fichier audio pour améliorer l'accessibilité. Au cas où un format n'est pas pris en charge par un navigateur, une version avec le format adapté est remplacée. Pour se faire, cette méthode requiert l'utilisation d'un élément spécialement créé pour : `<source>`. Cet élément accepte, en plus des attributs globaux, les attributs suivants :
  - **media** : spécifie le média pour lequel la source est optimisée. Les valeurs peuvent être `"all"`, `"tv"`, `"print"`, etc.
  - **src** : Spécifie l'URL du fichier audio.
  - **type** : Spécifie le type MIME du fichier audio.

(Ex :

```
<audio src="mamusique.mp3" controls preload="auto">
  <source src="mamusique.ogg" type="audio/ogg">
  <source src="mamusique.webm" type="audio/webm">
  Ce fichier ne peut pas être affiché.
```

`</audio>`). Cela joue d'abord le format mp3, si ce dernier n'est pas supporté, il essaie l'une des deux versions dans les `<source>`.

Dans tous les cas, cela produit :



## L'ELEMENT <VIDEO>

L'élément <video> permet d'intégrer une vidéo dans une page web. Il est souvent utilisé pour afficher des tutoriels, des bandes annonces, des démonstrations produits ou tout autre contenu visuel.

Il accepte, en dehors des attributs globaux, les attributs suivants :

1. **controls** : Affiche les contrôles du lecteur (lecture, pause, volume, plein écran, etc.).
2. **autoplay** : Démarre automatiquement la lecture dès que la page est chargée.
3. **loop** : Fait rejouer la vidéo automatiquement dès qu'elle se termine.
4. **muted** : Lance la vidéo en mode silencieux dès le chargement.
5. **mediagroup** : Permet de synchroniser plusieurs éléments multimédias (ex : plusieurs vidéos ou audios) afin qu'ils soient contrôlés ensemble (lecture, pause, etc.).
6. **crossorigin** : Définit comment le navigateur gère les requêtes vidéo provenant d'un autre domaine. Cet attribut prend les valeurs :
  - "anonymous" : Aucune information d'authentification n'est envoyée.
  - "use-credentials" : Envoie les cookies et identifiants avec la requête.
7. **preload** : Contrôle si la vidéo doit être préchargée ou non. Cet attribut prend les valeurs :
  - "auto" : Le navigateur décide du niveau de préchargement (par défaut).
  - "metadata" : Seules les métadonnées sont chargées (durée, dimensions...).
  - "none" : Rien n'est chargé tant que l'utilisateur n'interagit pas.
8. **poster** : Définit une image d'aperçu qui s'affiche avant que la lecture ne commence.
9. **width** et **height** : Spécifient les dimensions d'affichage de la vidéo (en pixels).
10. **playsinline** : Permet à la vidéo de se jouer directement dans la page, sans passer automatiquement en plein écran sur les appareils mobiles.
11. **src** : Spécifie directement l'URL du fichier vidéo si aucun élément <source/> n'est utilisé.

Il existe deux manières de charger une vidéo en HTML :

1. La méthode **mono-source** : Cette méthode consiste à ne fournir qu'une seule version du fichier vidéo à afficher, en spécifiant son URL dans l'attribut **src** (Ex :  
`<video src="mapresentation.mp4" controls poster="vignette.jpg">  
 Ce fichier ne peut pas être affiché.  
</video>`). Cette méthode est simple, mais comporte un inconvénient majeur : si le format de la vidéo n'est pas supporté par le navigateur, ce dernier ne pourra pas proposer d'alternative.
2. La méthode **multi-source** : Cette méthode consiste à fournir plusieurs versions d'un même fichier vidéo dans différents formats. Elle permet d'assurer une meilleure compatibilité entre navigateurs. Pour ce faire, on utilise aussi l'élément <source>. (Ex :  
`<video controls width="600" preload="auto" poster="vignette.jpg">  
 <source src="mapresentation.mp4" type="video/mp4">  
 <source src="mapresentation.webm" type="video/webm">  
 Ce fichier ne peut pas être affiché.  
</video>`). Cela joue d'abord le format mp4. Si ce dernier n'est pas supporté, il essaiera la version webm qui sera chargée.

Dans tous les cas, cela produit :

