

COURS HTML-CSS

Dans ce document sont présentées les principales notions en HTML et CSS abordées durant le cours. Bien entendu, le volume de connaissance associé à ces deux langages explique que ce document ne cherche nullement à être exhaustif.

Pour de plus amples informations, n'hésitez pas à consulter la documentation en ligne sur la Mozilla Developper Network (<https://developer.mozilla.org/fr/docs/Web>) ainsi que le site Can I Use (<https://caniuse.com/>) pour la compatibilité d'une balise HTML ou d'une propriété CSS avec le parc actuel des navigateurs.

Bonne lecture !

I - PREMIERS PAS

1) STRUCTURE DE BASE D'UNE PAGE WEB

Une page HTML est structurée de manière à ce que le navigateur puisse la lire et l'afficher correctement. Voici les éléments essentiels de cette structure :

- **Doctype** : La première ligne d'une page HTML. Elle indique au navigateur la version de HTML utilisée. Pour HTML5, c'est simplement `<!DOCTYPE html>`.
- **Balise `<html>`** : C'est la racine de votre document HTML. Elle englobe toutes les autres balises.
- **Balise `<head>`** : Cette section contient des informations sur le document, qui ne sont généralement pas affichées directement sur la page. Cela inclut le titre de la page, les liens vers des feuilles de style CSS, des scripts, et d'autres métadonnées.
- **Balise `<title>`** : Placée dans le `<head>`, elle définit le titre de la page, qui apparaît dans l'onglet du navigateur.
- **Balise `<body>`** : C'est ici que vous placez le contenu qui sera visible sur la page web. Tout ce qui est affiché dans la fenêtre du navigateur est placé à l'intérieur de cette balise.

Voici une page de base :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Titre de la Page</title>
5  </head>
6  <body>
7      <!-- Le contenu de la page sera ici --&gt;
8  &lt;/body&gt;
9  &lt;/html&gt;
</pre>

```

Cet exemple contient un commentaire dans la balise `<body>` (voir au point 7).

2) LA BALISE `<p>` POUR LES PARAGRAPHES

La balise `<p>` est utilisée pour définir un paragraphe dans une page web. Voici quelques points clefs :

- **Bloc de texte** : La balise `<p>` crée un bloc de texte. Chaque paragraphe est automatiquement séparé par un espace avant et après.
- **Usage** : Elle est utilisée pour regrouper des phrases ou des idées similaires, facilitant la lecture et l'organisation du contenu.
- **Formatage** : Par défaut, les paragraphes sont affichés avec une marge en haut et en bas. Vous pouvez modifier ce comportement avec CSS.

Un exemple avec la balise `<p>` :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemple de Paragraphes</title>
5  </head>
6  <body>
7      <p>Ceci est un paragraphe. Il contient du texte et
8          peut être long ou court.</p>
9      <p>Voici un autre paragraphe. Les paragraphes sont
10         séparés visuellement pour une meilleure lisibilité.</p>
11 </body>
12 </html>

```

Dans cet exemple, deux paragraphes sont définis. Chacun est entouré par une balise `<p>` et ils sont automatiquement espacés pour une lecture aisée.

3) LA BALISE BR

La balise `
` en HTML est utilisée pour insérer un saut de ligne dans le texte. Contrairement à la balise `<p>` qui crée un nouveau paragraphe avec un espace avant et après, `
` est utilisé pour un simple retour à la ligne sans espace supplémentaire. Voici quelques points clefs sur l'utilisation de cette balise :

- Saut de ligne simple** : `
` crée un saut de ligne sans commencer un nouveau paragraphe. C'est utile pour des adresses ou des poèmes où la structure de ligne est importante.
- Balise auto-fermante** : `
` est une balise auto-fermante, ce qui signifie qu'elle ne nécessite pas de balise de fermeture. Vous l'utilisez simplement comme `
` ou `
`.
- Pas de contenu** : Cette balise ne contient pas de contenu et est uniquement utilisée pour le formatage du texte.
- Utilisation avec prudence** : Son utilisation excessive peut rendre le texte difficile à lire et à maintenir. Il est souvent préférable d'utiliser des balises de paragraphe ou des éléments CSS pour contrôler l'espacement et la mise en page.

Voici un exemple simple montrant comment `
` peut être utilisé pour des retours à la ligne :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemple avec Balise br</title>
5  </head>
6  <body>
7      <p>
8          Ceci est un exemple de texte avec des sauts de ligne.<br>
9          Après cette phrase, il y aura un retour à la ligne.<br>
10         Et encore un autre ici.
11      </p>
12  </body>
13 </html>

```

Dans cet exemple, la balise `
` est utilisée deux fois à l'intérieur d'un paragraphe pour créer des sauts de ligne. Cela permet de séparer visuellement les phrases sans commencer un nouveau paragraphe.

4) LES TITRES

Les balises de titre en HTML, allant de `<h1>` à `<h6>`, sont utilisées pour structurer le contenu en termes de titres et de sous-titres, jouant un rôle crucial dans l'organisation et la hiérarchisation de l'information sur une page web.

Voici les caractéristiques des balises de titre `<h1>` à `<h6>` :

1. **Hiérarchie** : Les balises `<h1>` à `<h6>` représentent différents niveaux de titres, `<h1>` étant le plus important et `<h6>` le moins important. Cette hiérarchie aide à structurer le contenu de manière logique.
2. **Importance pour le SEO (Search Engine Optimization)** : Les moteurs de recherche utilisent ces balises pour comprendre la structure et le contenu d'une page. Un bon usage des balises de titre peut améliorer le référencement naturel (SEO) de la page.
3. **Style par défaut** : Chaque balise de titre a un style par défaut (taille de police, marge, etc.), mais cela peut être personnalisé avec CSS.
4. **Utilisation unique de `<h1>`** : Il est recommandé d'utiliser la balise `<h1>` une seule fois par page pour le titre principal. Cela aide à définir clairement le sujet de la page.
5. **Accessibilité** : Utiliser correctement ces balises améliore l'accessibilité du site, permettant aux lecteurs d'écran de naviguer efficacement dans le contenu.

Voici un exemple d'utilisation des balises de titre :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemple de Balises de Titre</title>
5  </head>
6  <body>
7      <h1>Titre Principal de la Page</h1>
8      <h2>Sous-titre Niveau 1</h2>
9      <p>Quelques informations sous le sous-titre niveau 1.</p>
10     <h3>Sous-titre Niveau 2</h3>
11     <p>Informations supplémentaires sous le sous-titre niveau 2.</p>
12     <h4>Sous-titre Niveau 3</h4>
13     <p>Détails sous le sous-titre niveau 3.</p>
14     <!-- Vous pouvez continuer avec h5 et h6 si nécessaire -->
15 </body>
16 </html>

```

Dans cet exemple, les balises de titre sont utilisées pour structurer le contenu de manière hiérarchique. Le `<h1>` définit le titre principal, suivi par des sous-titres de niveaux inférieurs (`<h2>`, `<h3>`, etc.). Cela crée une structure claire et facile à suivre pour les utilisateurs et les moteurs de recherche.

5) BALISE HR

La balise `<hr>` en HTML est utilisée pour créer une ligne horizontale dans la page, servant principalement de séparateur thématique entre différentes sections de contenu. Voici une explication détaillée de cette balise `<hr>` :

1. **Séparateur visuel** : `<hr>` est utilisé pour insérer une ligne de séparation horizontale, ce qui aide à diviser le contenu en sections distinctes visuellement.
2. **Balise auto-fermante** : Comme la balise `
`, `<hr>` est également une balise auto-fermante. Elle ne contient pas de contenu et ne nécessite pas de balise de fermeture. Vous l'utilisez simplement comme `<hr>` ou `<hr/>`.
3. **Signification sémantique** : En plus de sa fonction visuelle, `<hr>` peut également indiquer un changement de sujet ou une pause dans la narration dans le contenu du texte.
4. **Personnalisation avec CSS** : Bien que `<hr>` crée une ligne horizontale simple par défaut, son apparence peut être largement personnalisée avec CSS, y compris la couleur, la largeur, la marge, et même la transformation en d'autres formes décoratives.
5. **Utilisation moderne** : Dans les versions antérieures de HTML, `<hr>` était souvent utilisé pour des raisons purement esthétiques. Cependant, dans HTML5, son utilisation est encouragée principalement pour des raisons sémantiques et structurelles.

Un exemple d'utilisation de `<hr>` :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemple de Balises de Titre</title>
5  </head>
6  <body>
7      <h1>Titre Principal de la Page</h1>
8      <h2>Sous-titre Niveau 1</h2>
9      <p>Quelques informations sous le sous-titre niveau 1.</p>
10     <h3>Sous-titre Niveau 2</h3>
11     <p>Informations supplémentaires sous le sous-titre niveau 2.</p>
12     <h4>Sous-titre Niveau 3</h4>
13     <p>Détails sous le sous-titre niveau 3.</p>
14     <!-- Vous pouvez continuer avec h5 et h6 si nécessaire -->
15  </body>
16  </html>

```

Dans cet exemple, la balise `<hr>` est utilisée pour séparer visuellement deux sections différentes de contenu. Chaque section est introduite par un titre (`<h2>`) et suivie d'un paragraphe. La ligne horizontale créée par `<hr>` fournit une pause claire et une distinction entre ces sections.

6) LES HYPERLIENS

La balise `<a>` en HTML est utilisée pour créer des hyperliens, qui sont des éléments fondamentaux du web permettant de naviguer d'une page à une autre ou d'accéder à des ressources comme des documents ou des images. Cette balise peut être

enrichie avec divers attributs, dont **title** et **target**, qui ajoutent des fonctionnalités et des informations supplémentaires.

A - CARACTERISTIQUES DE LA BALISE <A>

1. **Création de liens** : La balise **<a>**, abréviation de “anchor” (ancre en français), est utilisée pour lier à d’autres pages ou ressources. Elle est essentielle pour la navigation web.
2. **Attribut href** : L’attribut le plus important de la balise **<a>** est **href** (Hypertext Reference), qui spécifie l’URL de la page ou de la ressource vers laquelle le lien doit mener.

B - ATTRIBUTS TITLE ET TARGET

1. **Attribut title** :
 - **Description** : Fournit des informations supplémentaires sur le lien. Cette information apparaît généralement comme une infobulle lorsque le curseur survole le lien.
 - **Accessibilité** : Utile pour l’accessibilité, car il donne un contexte supplémentaire, en particulier pour les utilisateurs qui utilisent des lecteurs d’écran.
 - **SEO (Search Engine Optimization)** : Peut contribuer au référencement, bien que son impact soit limité.
2. **Attribut target** :
 - **Contrôle de la navigation** : Détermine comment le lien sera ouvert. Par exemple, **target="_blank"** ouvre le lien dans un nouvel onglet ou une nouvelle fenêtre.
 - **Valeurs courantes** :
 - **_blank** : Ouvre le lien dans un nouvel onglet/fenêtre.
 - **_self** : Ouvre le lien dans le même cadre ou la même fenêtre (comportement par défaut).
 - **_parent** : Ouvre le lien dans le cadre parent.
 - **_top** : Ouvre le lien dans le cadre de niveau supérieur.

Voici un exemple d’utilisation de la balise **<a>**

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemple de Balise a</title>
5  </head>
6  <body>
7      <p>
8          Visitez <a href="https://www.example.com" title="Aller à Example.com"
9              target="_blank">Example.com</a> pour plus d'informations.
10     </p>
11 </body>
12 </html>

```

Dans cet exemple, le lien vers “Example.com” utilise la balise **<a>** avec les attributs **href**, **title** et **target**. L’attribut **title** fournit une courte description qui apparaîtra lorsque l’utilisateur survole le lien, et **target="_blank"** fait en sorte que le lien s’ouvre dans un nouvel onglet du navigateur.

7) LES COMMENTAIRES

Les commentaires en HTML sont utilisés pour ajouter des notes explicatives ou des rappels dans le code source, sans qu’ils soient affichés dans le navigateur. Les commentaires peuvent être extrêmement utiles pour documenter le code, en

particulier dans les projets collaboratifs ou lorsqu'un code est révisé après une longue période. Voici les détails sur l'utilisation des commentaires en HTML :

1. Syntaxe :

- Les commentaires en HTML commencent par `<!--` et se terminent par `-->`.
- Tout ce qui se trouve entre ces marqueurs est ignoré par le navigateur et n'est pas affiché sur la page web.

2. Utilisations courantes :

- **Explication du code** : Pour expliquer des parties spécifiques du code, ce qui le rend plus compréhensible pour les autres développeurs ou pour vous-même à l'avenir.
- **Désactivation temporaire de code** : Pour désactiver temporairement des parties du code HTML sans les supprimer complètement.
- **Notes et rappels** : Pour laisser des notes ou des rappels sur certaines parties du code ou sur des modifications à apporter.

3. Non visible pour les utilisateurs :

- Les commentaires ne sont pas visibles pour les utilisateurs dans le navigateur, mais ils peuvent être vus en affichant la source de la page (par exemple, en faisant un clic droit sur la page et en choisissant "Afficher le code source de la page").

4. Pas d'impact sur la performance :

- Les commentaires n'ont aucun impact sur la performance de la page, mais il est bon de les utiliser avec modération, surtout dans les fichiers de grande taille, pour maintenir la lisibilité du code.

Voici un exemple contenant deux commentaires :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemple de Commentaires HTML</title>
5  </head>
6  <body>
7      <!-- Ceci est un commentaire en HTML. Il n'apparaîtra
8          pas dans le navigateur. -->
9      <p>Ceci est un paragraphe visible sur la page.</p>
10     <!-- Vous pouvez utiliser des commentaires pour
11         expliquer des parties spécifiques du code -->
12  </body>
13  </html>

```

Dans cet exemple, deux commentaires sont inclus. Ils expliquent ce que fait le code ou laissent une note, mais ne sont pas affichés dans le navigateur. Les commentaires sont un excellent moyen de rendre votre code plus lisible et de communiquer des informations importantes aux autres développeurs ou à vous-même pour des références futures.

8) LES IMAGES

La balise `` en HTML est utilisée pour intégrer des images dans une page web. C'est un élément essentiel pour ajouter des visuels et améliorer l'expérience utilisateur sur un site web.

Voici les caractéristiques de la balise `` :

1. **Intégration d'images** : La balise `` permet d'insérer des images dans le contenu HTML. Elle peut être utilisée pour afficher des images de différents formats, tels que JPEG, PNG, GIF, SVG, etc.

2. **Balise auto-fermante :** est une balise auto-fermante, ce qui signifie qu'elle ne nécessite pas de balise de fermeture. Elle est utilisée sous la forme .
3. **Attributs essentiels :**
 - **src (source)** : Cet attribut est obligatoire et spécifie le chemin vers l'image que vous souhaitez afficher.
 - **alt (texte alternatif)** : Cet attribut fournit une description textuelle de l'image pour l'accessibilité et en cas où l'image ne peut pas être chargée.
4. **Autres attributs courants :**
 - **title** : Fournit des informations supplémentaires qui apparaissent généralement comme une infobulle lorsque le curseur survole l'image.
 - **width et height** : Permettent de spécifier la largeur et la hauteur de l'image. Il est recommandé de spécifier ces dimensions pour améliorer le rendu de la page lors du chargement.
5. **Importance pour l'accessibilité** : L'attribut **alt** est crucial pour les utilisateurs qui dépendent des lecteurs d'écran pour naviguer sur Internet, car il décrit ce que l'image représente.
6. **Optimisation des performances** : Il est important de veiller à la taille des fichiers d'image pour ne pas ralentir le temps de chargement de la page.

Un exemple d'utilisation de la balise :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Exemple de Balise img</title>
5  </head>
6  <body>
7      
13     <p>Ci-dessus se trouve une image avec une description,
14     un titre, et des dimensions spécifiées.</p>
15 </body>
16 </html>
```

Dans cet exemple, la balise est utilisée pour intégrer une image avec les attributs **src**, **alt**, **title**, **width** et **height**. L'attribut **src** indique le chemin de l'image, **alt** fournit une description textuelle pour l'accessibilité, **title** offre des informations supplémentaires lors du survol de l'image, et **width** et **height** contrôlent les dimensions de l'image affichée.

9) INTRODUCTION AU CSS

Le CSS (Cascading Style Sheets, feuilles de style en cascade) est un langage de feuille de style utilisé pour décrire la présentation d'un document écrit en HTML ou XML (y compris les langages basés sur XML comme SVG ou XHTML). Le CSS décrit la façon dont les éléments doivent être rendus à l'écran, sur papier, dans la parole, ou sur d'autres médias.

A - GRANDES CARACTERISTIQUES DU CSS

Parmi les caractéristiques du CSS, on peut noter les trois suivantes :

1. **Séparation du contenu et de la présentation** : L'un des principaux avantages de CSS est qu'il permet une séparation claire entre le contenu d'une page (HTML) et sa présentation (style). Cela rend le développement et la maintenance des sites web plus faciles et plus efficaces.
2. **Contrôle de la mise en page** : Avec CSS, vous pouvez contrôler la mise en page de plusieurs pages web simultanément. Les styles peuvent être appliqués à des éléments spécifiques, à des groupes d'éléments, ou à des pages entières.
3. **Flexibilité** : CSS offre une grande flexibilité dans la personnalisation des styles, permettant de modifier la taille, la couleur, la marge, la bordure, et bien d'autres propriétés des éléments HTML.

B - FAÇONS D'INTEGRER LE CSS DANS UN DOCUMENT HTML

Il existe trois méthodes principales pour appliquer des styles CSS à un document HTML :

1. **CSS inline (directement au niveau de la balise ciblée)** :
 - Le style est appliqué directement sur les éléments HTML individuels via l'attribut **style** : `<p style="color: blue;*>Ce texte sera en bleu.</p>`
 - Type d'usage : Pratique pour des modifications rapides ou des tests, mais pas recommandé pour le style de sites entiers car cela rend le code difficile à maintenir.
2. **Balise `<style>` dans l'en-tête** :
 - Les styles sont placés dans une balise **<style>** dans la section **<head>** du document HTML :

```

1  <head>
2    <style>
3      p { color: blue; }
4    </style>
5  </head>

```

3. **Feuille de Style Externe (External CSS)** :
 - Type d'usage : Utile pour des styles spécifiques à une seule page. Cela garde les styles séparés du contenu HTML mais dans le même fichier.

- Les styles sont écrits dans un fichier séparé avec l'extension **.css**, puis liés au document HTML via la balise **<link>** :

```

1  <head>
2    <link rel="stylesheet" href="styles.css">
3  </head>

```

- Type d'usage : C'est la méthode recommandée pour styliser des sites web complets. Elle permet de maintenir la cohérence sur plusieurs pages et facilite la maintenance.

Chaque méthode a ses propres avantages et cas d'utilisation. En pratique, pour des projets de plus grande envergure, l'utilisation de feuilles de style externes est généralement privilégiée pour sa maintenabilité et sa flexibilité.

10) CLASS ET ID

Les attributs **class** et **id** en HTML sont utilisés pour identifier des éléments spécifiques dans un document, permettant ainsi de cibler et de styliser ces éléments avec CSS. Ils jouent un rôle crucial dans la structuration et la mise en forme des pages web.

A - L'ATTRIBUT **CLASS** EN HTML ET CSS

1. Utilisation en HTML :

- L'attribut **class** peut être appliqué à n'importe quel élément HTML.
- Il permet de regrouper plusieurs éléments sous une même classe pour appliquer un style commun.
- Un élément peut avoir plusieurs classes, séparées par des espaces.

```
1 <div class="box highlight">Contenu</div>
```

2. Ciblage en CSS :

- En CSS, les classes sont ciblées en utilisant un point **.** suivi du nom de la classe.
- Les styles définis pour une classe s'appliquent à tous les éléments HTML qui ont cette classe.

```
1 .box {
2   border: 1px solid black;
3 }
4
5 .highlight {
6   background-color: yellow;
7 }
```

B - ATTRIBUT **ID** EN HTML ET CSS

1. Utilisation en HTML :

- L'attribut **id** est unique dans un document HTML, ce qui signifie qu'un seul élément peut avoir un **id** particulier.
- Il est utilisé pour identifier un élément spécifique, ce qui est utile pour le ciblage avec JavaScript et pour la navigation interne dans la page (avec des ancrages).

```
1 <div id="header">En-tête</div>
```

2. Ciblage en CSS :

- En CSS, les **id** sont ciblés en utilisant un dièse **#** suivi du nom de l'**id**.
- Comme un **id** est unique, les styles appliqués à un **id** ne concernent que cet élément spécifique.

```
1 #header {
2   background-color: blue;
3   color: white;
4 }
```

C - DIFFÉRENCES ET CHOIX ENTRE **CLASS** ET **ID**

- **Réutilisabilité** : Les classes sont destinées à être réutilisées sur plusieurs éléments, tandis qu'un **id** est unique à un seul élément.

- **Spécificité en CSS :** En CSS, les **id** ont une spécificité plus élevée que les classes. Cela signifie que si un élément est ciblé à la fois par un **id** et une classe, les styles définis par l'**id** auront la priorité, sauf si la spécificité de la classe est augmentée (par exemple, en utilisant des sélecteurs plus complexes ou en utilisant **!important**).
- **Choix :** Utilisez **class** lorsque vous avez besoin de styliser plusieurs éléments de manière similaire. Utilisez **id** pour cibler un élément spécifique, pour le JavaScript, ou pour la navigation interne dans la page.

D - EN BREF

Class et **id** sont des outils puissants pour organiser et styliser les éléments HTML. Le choix entre les deux dépend de la nécessité de réutiliser les styles ou de cibler des éléments spécifiques.

11) BALISES STRONG ET EM

Les balises **** et **** en HTML sont utilisées pour donner une emphase particulière au texte. Elles ont également une certaine importance en SEO (Search Engine Optimization), bien que leur impact soit plus subtil et contextuel que d'autres facteurs SEO.

A - BALISE

1. **Utilisation :**
 - La balise **** est utilisée pour indiquer que le texte est d'une importance particulière.
 - Visuellement, le texte entouré par **** apparaît en gras.
2. **Importance en SEO :**
 - Utiliser **** peut aider les moteurs de recherche à comprendre quels mots ou phrases dans votre contenu sont particulièrement importants.
 - Cela dit, son impact direct sur le classement SEO est limité. L'abus de cette balise peut diluer son efficacité et être perçu comme du bourrage de mots-clés.

B - BALISE

1. **Utilisation :**
 - La balise **** est utilisée pour mettre l'accent sur un texte, indiquant une intonation ou un stress dans la parole.
 - Visuellement, le texte entouré par **** apparaît en italique.
2. **Importance en SEO :**
 - Tout comme ****, **** peut être utilisé pour indiquer aux moteurs de recherche qu'une partie du texte est significative.
 - L'effet sur le SEO est similaire à celui de **** : utile pour souligner certains mots, mais avec un impact limité sur le classement global.

C - BONNES PRATIQUES

- **Usage contextuel :** Utilisez **** et **** de manière contextuelle pour mettre en valeur des mots ou des phrases qui sont vraiment importants dans le contexte de votre contenu.
- **Éviter l'excès :** N'utilisez pas ces balises de manière excessive. Leur utilisation doit être justifiée par le contenu et l'intention du texte.
- **Complément et non substitut :** Ces balises doivent compléter votre stratégie de contenu et de mots-clés, et non la remplacer. Le contenu de qualité et pertinent reste le facteur le plus important en SEO.

D - CONCLUSION

**** et **** sont utiles pour l'emphase textuelle et peuvent avoir un impact sur le SEO en aidant à clarifier quels aspects du contenu sont les plus pertinents. Cependant, leur utilisation doit être équilibrée et justifiée par le contexte du contenu.

12) BALISES DIV ET SPAN

Les balises **<div>** et **** sont deux éléments fondamentaux en HTML utilisés pour grouper et styliser des parties de votre document. Bien qu'ils soient similaires dans leur fonction de regroupement, ils ont des caractéristiques différentes qui influencent la manière dont ils sont utilisés dans la structure d'une page web.

A - BALISE <DIV>

1. Bloc de Niveau :

- **<div>** est un élément de type block. Cela signifie qu'il crée une "boîte" ou un bloc sur la page.
- Les éléments de niveau bloc commencent généralement sur une nouvelle ligne et s'étendent sur toute la largeur disponible, créant ainsi un bloc distinct.

2. Utilisation :

- Utilisé pour regrouper de grands blocs de contenu ou d'éléments.
- Très utile pour la mise en page, car il peut être stylisé avec CSS et est souvent utilisé avec des classes ou des id pour appliquer des styles spécifiques.

3. Exemple :

```

1  <div class="header">
2      <h1>Titre de la Page</h1>
3      <p>Introduction à la page...</p>
4  </div>

```

B - BALISE

1. Niveau en Ligne :

- **** est un élément de type inline. Il est utilisé pour regrouper ou appliquer des styles à une partie du texte sans provoquer de saut de ligne.
- Les éléments de niveau en ligne ne commencent pas sur une nouvelle ligne et prennent juste assez de place pour leur contenu.

2. Utilisation :

- Idéal pour styliser des portions de texte à l'intérieur d'autres éléments (comme des paragraphes) sans affecter le flux du document.
- Peut être utilisé pour appliquer des styles CSS ou pour identifier des parties du texte avec des classes ou des id.

3. Exemple :

```

1 <p>Ceci est un
2   <span class="highlight">texte important</span>
3   dans le paragraphe.
4 </p>
```

C - COMMENT CHOISIR ?

- **<div>** : Idéal pour structurer et organiser des sections plus larges de la page; élément de niveau bloc.
- **** : Meilleur pour le style et le ciblage de petites portions de texte ou d'autres éléments en ligne; élément de niveau en ligne.

En choisissant entre **<div>** et ****, la décision dépend principalement de la structure de votre contenu et de la manière dont vous souhaitez appliquer les styles CSS. **<div>** est plus adapté pour organiser la mise en page et créer des sections distinctes, tandis que **** est utile pour appliquer des styles spécifiques à des parties plus petites du contenu.

13) LE FLUX DANS UN DOCUMENT ET LA DIFFERENCE INLINE/BLOCK

La notion de flux dans un document HTML est un concept clé pour comprendre comment les éléments sont disposés et affichés sur une page web. Le "flux" se réfère à l'ordre et à la manière dont les éléments de bloc et en ligne sont placés et interagissent dans le document. La propriété CSS **display** joue un rôle crucial dans la manipulation de ce flux.

A - FLUX NORMAL DU DOCUMENT

1. Flux de Base :

- Dans le flux normal (aussi appelé "flux de document"), les éléments de type bloc s'empilent verticalement, l'un après l'autre, commençant sur une nouvelle ligne et s'étendant sur toute la largeur disponible.
- Les éléments en ligne sont disposés horizontalement, apparaissant dans le flux normal du texte. Ils ne commencent pas sur une nouvelle ligne et prennent juste assez de place pour leur contenu.

2. Impact sur la Mise en Page :

- Ce comportement par défaut détermine la structure de base d'une page web avant toute modification de style via CSS.

B - MODIFICATION DU FLUX AVEC DISPLAY

La propriété **display** permet de changer la façon dont les éléments interagissent avec ce flux normal :

1. Bloc à En Ligne :

- **display: inline;** fait qu'un élément de type bloc se comporte comme un élément en ligne, le faisant intégrer le flux de texte sans commencer sur une nouvelle ligne.
- Exemple : **div { display: inline; }** permet à un **<div>** de se comporter comme un élément en ligne.

2. En Ligne à Bloc :

- **display: block;** transforme un élément en ligne en élément de type bloc, le faisant occuper toute la largeur disponible et commencer sur une nouvelle ligne.
- Exemple : **span { display: block; }** fait qu'un **** se comporte comme un élément bloc.

3. Inline-Block :

- **display: inline-block;** permet à un élément de se placer dans le flux en ligne tout en acceptant des propriétés de dimensionnement comme un élément de type bloc.

4. Retrait du Flux :

- **display: none;** retire complètement l'élément du flux, le rendant invisible et non pris en compte dans le layout.

C - AUTRES PROPRIETES INFLUANT SUR LE FLUX

- **float** : Permet à un élément de “flotter” à gauche ou à droite, affectant la manière dont le texte et les autres éléments s’écoulent autour de lui. Historiquement, il a été utilisé pour faire de la mise en page ce qui n’était pas son usage à l’origine, mais cet utilisation est, depuis plusieurs années, abandonnée et fortement déconseillée.
- **position** : Cette propriété modifie la position d’un élément dans le flux. Par exemple, **position: absolute;** retire l’élément du flux normal et le positionne par rapport à son conteneur le plus proche positionné.

D - CONCLUSION

La compréhension du flux normal du document et la manière dont les propriétés CSS comme **display**, **float**, et **position** affectent ce flux est essentielle pour maîtriser la mise en page en HTML et CSS. Cela permet aux développeurs de créer des mises en page complexes et réactives, adaptées aux besoins spécifiques de leurs projets web.

II - AMELIORATION 2

1) LE CIBLAGE EN CSS

En CSS, il existe plusieurs façons de cibler des éléments pour appliquer des styles. Chaque type de sélecteur a une fonction spécifique et permet de sélectionner des éléments en fonction de leur relation dans la structure du document HTML. Voici une explication des sélecteurs les plus courants :

A. SELECTION PAR IDENTIFIANT

- **Syntaxe : #identifiant**
- **Fonction** : Sélectionne tous l’élément qui a l’identifiant spécifiée. Attention, cet identifiant est toujours unique dans le document HTML.
- **Exemple :**

```
1  #important {
2      font-weight: bold;
3 }
```

Ici, tous l’élément avec l’identifiant **important** sera en gras.

B. SELECTION PAR CLASSE

- **Syntaxe : .classname**
- **Fonction** : Sélectionne tous les éléments qui ont la classe spécifiée.

- **Exemple :**

```

1 .alert {
2   color: red;
3 }
```

Ici, tous les éléments avec la classe **alert** auront le texte en rouge.

C. SELECTEUR DESCENDANT

- **Syntaxe :** A B
- **Fonction :** Sélectionne tous les éléments B qui sont des descendants de A (c'est-à-dire, B est à l'intérieur de A, directement ou indirectement).
- **Exemple :**

```

1 div p {
2   color: red;
3 }
```

Ici, tous les **<p>** à l'intérieur d'une **<div>** seront colorés en rouge.

D. SELECTEUR ENFANT

- **Syntaxe :** A > B
- **Fonction :** Sélectionne tous les éléments B qui sont des enfants directs de A.
- **Exemple :**

```

1 ul > li {
2   font-weight: bold;
3 }
```

Ici, seuls les **** qui sont des enfants directs d'un **** seront en gras.

E. SELECTEUR DE VOISIN DIRECT

- **Syntaxe :** A + B
- **Fonction :** Sélectionne l'élément B qui suit immédiatement A, à condition qu'ils aient le même parent.

- **Exemple :**

```
1  h1 + p {
2    font-size: 18px;
3 }
```

Ici, seul le premier `<p>` qui suit directement un `<h1>` aura une taille de police de 18px.

F. SELECTION PAR NOM D'ELEMENT

- **Syntaxe :** `element`
- **Fonction :** Sélectionne tous les éléments de ce type.
- **Exemple :**

```
1  p {
2    line-height: 1.5;
3 }
```

Ici, tous les paragraphes (`<p>`) auront un interligne de 1.5.

G - CONCLUSION

Chaque type de sélecteur a un rôle spécifique et peut être utilisé seul ou en combinaison pour cibler précisément les éléments souhaités dans le document HTML. La compréhension de ces sélecteurs est essentielle pour écrire des styles CSS efficaces et bien structurés.

2) ANCRE ET LIEN VERS UN MAIL

L'utilisation d'ancres dans une page HTML et les liens `mailto:` sont deux fonctionnalités importantes pour améliorer la navigation et la communication sur les sites web.

A - ANCRES DANS UNE PAGE HTML

Les ancrées sont utilisées pour créer des liens vers différentes parties d'une même page ou vers des sections spécifiques d'autres pages. Elles sont particulièrement utiles pour les longs documents, permettant aux utilisateurs de sauter directement à l'information pertinente. Pour créer une ancre, il faut:

1. **Définir l'Ancre :**
 - Utilisez l'attribut `id` sur n'importe quel élément HTML pour marquer la destination de l'ancré.
 - Exemple : `<h2 id="section1">Section 1</h2>`
2. **Lien vers l'Ancre :**
 - Créez un lien en utilisant la balise `<a>` avec l'attribut `href` pointant vers l'`id` de l'ancré précédée d'un dièse (#).
 - Exemple : `Aller à la Section 1`

Ceci nous donne, par exemple :

```

1 <a href="#section1">Aller à la Section 1</a>
2
3 ...
4
5 <h2 id="section1">Section 1</h2>
6 <p>Contenu de la section 1...</p>
```

Dans cet exemple, en cliquant sur le lien “Aller à la Section 1”, l’utilisateur sera amené directement à l’élément **<h2>** qui a l’**id="section1"**.

B - LIENS MAILTO

Les liens **mailto:** sont utilisés pour créer des liens qui, lorsqu’ils sont cliqués, ouvrent le client de messagerie par défaut de l’utilisateur avec une adresse e-mail pré-remplie. Pour créer un lien mailto :

- **Syntaxe de base :** `Envoyer un Email`
- **Avec sujet :** Vous pouvez ajouter un sujet à l’email en utilisant `?subject=`.
 - Exemple : `Envoyer un Email avec Sujet`
- **Avec corps de message :** Ajoutez un corps de message avec `&body=`.
 - Exemple : `Envoyer un Email Complet`

Ceci nous donne, par exemple :

```

1 <a href="mailto:exemple@domaine.com">Envoyer un Email</a>
2 <a href="mailto:exemple@domaine.com?subject=Nouvelles Informations&body=Bonjour, voici les dernières nouvelles...">
3   Envoyer un Email avec Sujet et Corps
4 </a>
5
```

Dans cet exemple, le premier lien ouvre le client de messagerie avec l’adresse e-mail préremplie, tandis que le second inclut également un sujet et un corps de message prédéfinis.

C - CONCLUSION

Les ancrées et les liens **mailto:** sont des outils essentiels en HTML pour améliorer la navigation dans les documents et faciliter la communication par e-mail. Ils sont simples à mettre en place et peuvent considérablement améliorer l’expérience utilisateur sur votre site web.

3) REPRESENTER UN CONTENU, <FIGURE> ET <FIGCAPTION>

Les balises **<figure>** et **<figcaption>** en HTML5 sont utilisées pour représenter un contenu indépendant, souvent une image, un diagramme, un code source, ou une illustration, accompagné d'une légende ou d'une explication. Ces balises aident à structurer le contenu de manière sémantique, rendant les documents plus accessibles et compréhensibles.

A - UTILISATION DE <FIGURE>

1. But :

- La balise **<figure>** est utilisée pour encapsuler des médias, tels que des images, des vidéos, des graphiques, du code, etc., ainsi que leur légende (**<figcaption>**).
- Elle représente un contenu autonome, ce qui signifie que si vous le déplacez ailleurs dans le document, il conserve son sens.

2. Structure :

- **<figure>** peut contenir plusieurs types de contenu, mais est souvent utilisé avec des images.
- Il peut être placé indépendamment du flux principal du texte et est généralement utilisé pour des éléments qui sont référencés depuis le contenu principal.

B - UTILISATION DE <FIGCAPTION>

1. But :

- **<figcaption>** fournit une légende ou une description pour le contenu de **<figure>**.
- Il est facultatif, mais lorsqu'il est utilisé, il doit être placé comme premier ou dernier enfant de **<figure>**.

2. Relation avec <figure> :

- **<figcaption>** est associé à **<figure>** pour fournir un contexte additionnel. Cela aide les lecteurs et les technologies d'assistance à comprendre la relation entre l'image (ou autre contenu) et le texte qui l'accompagne.

C - EXEMPLE D'UTILISATION DE CES DEUX BALISES

```

1  <figure>
2    
6    <figcaption>Légende expliquant l'image.</figcaption>
7  </figure>

```

Dans cet exemple, une image est incluse à l'intérieur d'une balise **<figure>**, avec une légende fournie par **<figcaption>**. La légende décrit et complète l'image, fournissant un contexte supplémentaire.

D - BONNES PRATIQUES

- **Accessibilité** : Utilisez toujours l'attribut **alt** avec des images à l'intérieur de **<figure>** pour assurer l'accessibilité.
- **Contexte** : Utilisez **<figcaption>** pour fournir des informations supplémentaires qui aident à comprendre le contenu de **<figure>**.
- **Indépendance** : Placez des éléments dans **<figure>** qui sont autonomes et peuvent être déplacés dans le document sans perdre leur signification.

E - EN BREF

<figure> et **<figcaption>** sont des outils utiles pour présenter des contenus visuels et autres médias de manière sémantique et structurée, améliorant ainsi l'accessibilité et la compréhension du contenu sur les pages web.

4) MISE EN FORME DU TEXTE EN CSS

Mettre en forme du texte en CSS est essentiel pour créer un design web attrayant et lisible. Voici un aperçu des propriétés CSS que vous avez mentionnées, qui sont toutes utilisées pour styliser le texte :

1. **font-size** : Définit la taille de la police de caractères.

```
1  p {
2      font-size: 16px;
3 }
```

Ici, la taille de la police pour les paragraphes est définie à 16 pixels.

2. **font-style** : Définit le style de la police, comme normal, italique ou oblique.

```
1  em {
2      font-style: italic;
3 }
```

Ici, le texte dans **** sera en italique.

3. **font-weight** : Définit l'épaisseur de la police, comme normal, bold, ou en utilisant des valeurs numériques (100 à 900).

```
1  strong {
2      font-weight: bold;
3 }
```

Ici, le texte dans **** sera en gras.

4. **text-align** : Définit l'alignement horizontal du texte, comme left, right, center, ou justify.

```
1  h1 {
2      text-align: center;
3 }
```

Ici, le texte dans **<h1>** sera centré.

5. **line-height** : Définit la hauteur de ligne du texte. Peut être un nombre, un pourcentage, ou une valeur comme **normal**.

```

1  p {
2    line-height: 1.5;
3 }
```

Ici, la hauteur de ligne pour les paragraphes est définie à 1.5 fois la taille de la police.

6. **text-transform**

- **Description** : Contrôle la capitalisation du texte, comme uppercase, lowercase, ou capitalize.

```

1  h2 {
2    text-transform: uppercase;
3 }
```

Ici, le texte dans **<h2>** sera transformé en majuscules.

7. **letter-spacing** : Définit l'espacement entre les caractères du texte.

```

1  p {
2    letter-spacing: 2px;
3 }
```

Ici, il y aura un espacement de 2 pixels entre chaque lettre dans les paragraphes.

8. **text-decoration** : définit la décoration du texte, comme underline, overline, line-through, ou none.

```

1  a {
2    text-decoration: underline;
3 }
```

Ici, les liens (**<a>**) seront soulignés.

Chacune de ces propriétés offre un contrôle fin sur l'apparence du texte, permettant aux concepteurs de créer des expériences utilisateur riches et visuellement attrayantes. En les combinant de manière créative, vous pouvez améliorer considérablement l'esthétique et la lisibilité de votre contenu web.

5) LES LISTES EN HTML ET LEUR MISE EN FORME EN CSS

En HTML, il existe deux types principaux de listes : les listes non ordonnées (unordered list : ``) et les listes ordonnées (ordered list : ``). Chacune utilise des éléments `` (list item) pour les entrées de la liste. Voici un aperçu de ces deux types de listes et des propriétés CSS couramment utilisées pour les mettre en forme.

A - LISTES NON ORDONNEES (``)

- **Usage** : Utilisées pour des listes où l'ordre des éléments n'est pas important.
- **Marqueurs par Défaut** : Les éléments de la liste (``) sont généralement précédés de puces.

```

1  <ul>
2    <li>Élément 1</li>
3    <li>Élément 2</li>
4    <li>Élément 3</li>
5  </ul>

```

B - LISTES ORDONNEES (``)

- **Usage** : Utilisées pour des listes où l'ordre des éléments est important.
- **Marqueurs par Défaut** : Les éléments de la liste sont généralement numérotés.

```

1  <ol>
2    <li>Premier élément</li>
3    <li>Deuxième élément</li>
4    <li>Troisième élément</li>
5  </ol>

```

C - PROPRIETES CSS POUR LA MISE EN FORME DES LISTES

1. **`list-style-type`** :
 - Définit le type de marqueur pour les listes.
 - Valeurs courantes : `disc`, `circle`, `square` pour ``; `decimal`, `lower-alpha`, `upper-roman`, etc., pour ``.
2. **`list-style-position`** :
 - Définit si les marqueurs sont à l'intérieur ou à l'extérieur du flux de contenu.
 - Valeurs : `inside`, `outside`.
3. **`list-style-image`** :
 - Remplace le marqueur standard par une image.
 - Valeur : URL de l'image (`url('chemin/vers/image.png')`).
4. **`padding` et `margin`** :
 - Ajuste l'espacement autour et à l'intérieur des listes.
 - Utile pour contrôler l'indentation des listes.
5. **`color` et `font-family`** :
 - Applique des styles de couleur et de police aux éléments de la liste.

D - MISE EN ŒUVRE

```

1  ul {
2      list-style-type: square;
3      list-style-position: inside;
4      padding-left: 20px;
5  }
6
7  ol {
8      list-style-type: upper-roman;
9      margin-left: 20px;
10 }
11
12 li {
13     color: blue;
14     font-family: Arial, sans-serif;
15 }
```

Dans cet exemple, les listes non ordonnées (``) ont des marqueurs carrés et sont indentées, tandis que les listes ordonnées (``) utilisent des chiffres romains majuscules et ont un margin à gauche. Les éléments de liste (``) sont stylisés avec une couleur bleue et une police Arial.

En utilisant ces propriétés CSS, vous pouvez personnaliser l'apparence des listes dans vos documents HTML pour les adapter au style et au design de votre site web.

III - AMELIORATION 2

1) UTILISATION DE POLICES

L'utilisation de différentes polices de caractères dans un document web peut être réalisée de plusieurs manières, notamment en utilisant une police installée sur le poste de l'utilisateur, en téléchargeant une police depuis le serveur avec `@font-face`, ou en utilisant une police de Google Fonts.

A - UTILISATION D'UNE POLICE DU POSTE DE L'UTILISATEUR

Dans ce cas, vous spécifiez une police qui est couramment installée sur la plupart des systèmes des utilisateurs. Il est conseillé de fournir des alternatives en cas où la police principale n'est pas disponible.

```

1  body {
2      font-family: Arial, Helvetica, sans-serif;
3  }
```

Ici, **Arial** est la police principale. Si **Arial** n'est pas disponible, le navigateur essaiera **Helvetica**, et s'il n'est pas disponible non plus, il utilisera la police sans-serif par défaut du système.

B - UTILISATION DE @FONT-FACE POUR DES POLICES TELECHARGEES

Avec **@font-face**, vous pouvez définir une police personnalisée en téléchargeant un fichier de police depuis votre serveur. Assurez-vous d'avoir les droits nécessaires pour utiliser et distribuer la police.

```

1  @font-face {
2      font-family: 'MaSuperPolice';
3      src: url('MaSuperPolice.woff2') format('woff2'),
4          url('MaSuperPolice.woff') format('woff');
5  }
6
7  body {
8      font-family: 'MaSuperPolice', Arial, sans-serif;
9  }
```

Dans cet exemple, **MaSuperPolice** est une police personnalisée que vous avez téléchargée sur votre serveur. Les formats **woff2** et **woff** sont utilisés pour une meilleure compatibilité entre les navigateurs.

C - UTILISATION D'UNE GOOGLE FONT

Google Fonts offre une large gamme de polices accessibles gratuitement. Vous pouvez les intégrer directement dans votre CSS via un lien ou en utilisant **@import**.

```

1  @import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');
2
3  body {
4      font-family: 'Roboto', sans-serif;
5  }
```

Dans cet exemple, la police **Roboto** est importée depuis Google Fonts. Vous pouvez spécifier les poids de la police (par exemple, 400 pour normal et 700 pour gras) dans l'URL.

D - CONCLUSION

Chaque méthode a ses avantages et ses inconvénients. L'utilisation de polices système garantit une grande vitesse de chargement et une bonne compatibilité, tandis que **@font-face** et Google Fonts offrent plus de personnalisation et de style à votre site web. Il est important de toujours prévoir des polices de secours pour assurer une bonne expérience utilisateur, quel que soit le support ou la disponibilité des polices choisies.

2) LES DIMENSIONS D'UN BLOC ET LES UNITES

La gestion des dimensions d'un bloc en HTML et CSS est cruciale pour contrôler l'apparence et la mise en page d'une page web. Les propriétés **width** et **height** sont les outils de base pour définir la largeur et la hauteur des éléments de type bloc. Voici comment elles fonctionnent et quelques considérations importantes lors de leur utilisation.

A - UTILISATION DE **WIDTH** ET **HEIGHT**

1. Définition de Base :

- **width** définit la largeur d'un élément.
- **height** définit la hauteur d'un élément.

2. Syntaxe :

```

1 .element {
2   width: 300px; /* Largeur de 300 pixels */
3   height: 150px; /* Hauteur de 150 pixels */
4 }
5

```

- Les valeurs peuvent être en pixels (**px**), en pourcentages (**%**), en **em**, **rem**, et d'autres unités CSS.

3. Largeur et Hauteur en Pourcentage :

- Lorsque **width** ou **height** sont définis en pourcentages, ils sont calculés par rapport à la taille du conteneur parent.
- Exemple : **width: 50%**; signifie que l'élément occupera 50% de la largeur de son conteneur parent.

B - CONSIDERATIONS IMPORTANTES

1. Box Model :

- En CSS, le modèle de boîte (box model) détermine comment les dimensions d'un élément sont calculées.
- Il comprend la largeur/hauteur (**width/height**), la marge (**margin**), la bordure (**border**), et le remplissage (**padding**).
- Par défaut, la largeur et la hauteur totales d'un élément incluent seulement le contenu, mais pas le padding, la bordure, ou la marge.

2. Box-sizing :

- La propriété **box-sizing** permet de changer la façon dont la largeur et la hauteur sont calculées.
- **box-sizing: content-box;** (valeur par défaut) signifie que **width** et **height** ne comprennent que le contenu.
- **box-sizing: border-box;** fait en sorte que **width** et **height** incluent le contenu, le padding et la bordure, mais pas la marge. Cela est souvent plus intuitif et facilite la gestion des dimensions.

3. Responsive Design :

- Utiliser des unités relatives comme les pourcentages ou **em** pour **width** et **height** peut rendre le design plus adaptable à différentes tailles d'écran.
- Cela est particulièrement important pour le design web réactif (responsive design).

C - LES UNITES EN CSS

Les unités en CSS sont essentielles pour définir la taille, l'espace, et d'autres propriétés des éléments. Parmi les unités les plus couramment utilisées, on trouve les pixels (**px**), les pourcentages (**%**), les **em** et les **rem**. Chacune de ces unités a des caractéristiques spécifiques qui les rendent utiles dans différents contextes.

1. Pixels (px) :
 - **Description** : Le pixel est l'unité de mesure la plus basique en CSS. Un pixel correspond à un point de l'écran.
 - **Utilisation** : Idéal pour des dimensions précises et fixes. Les pixels sont souvent utilisés pour définir des tailles de bordure, des ombres, et d'autres petites dimensions.
 - **Exemple** : `font-size: 16px;`
2. Pourcentages (%) :
 - **Description** : Les pourcentages sont relatifs à un autre élément, souvent l'élément parent. Par exemple, une largeur définie en pourcentage sera une fraction de la largeur de l'élément parent.
 - **Utilisation** : Très utile pour le design responsive, car les dimensions s'adaptent en fonction de la taille de l'élément parent ou de la fenêtre du navigateur.
 - **Exemple** : `width: 50%;`
3. EM
 - **Description** : L'unité **em** est relative à la taille de la police (**font-size**) de l'élément parent. Par exemple, **2em** signifie deux fois la taille de la police de l'élément parent.
 - **Utilisation** : Utile pour les mises en page qui doivent s'adapter à la taille de la police. Souvent utilisé pour les espacements (marges, padding) et les tailles de police.
 - **Exemple** : `font-size: 1.5em;`
4. REM
 - **Description** : L'unité **rem** (Root EM) est similaire à **em**, mais elle est toujours relative à la taille de la police de l'élément racine du document (**html**), et non à celle de l'élément parent.
 - **Utilisation** : Idéal pour un design cohérent sur toute la page, car elle permet d'éviter les problèmes de cascade des **em**. Souvent utilisée pour les tailles de police, les espacements, et les dimensions des éléments.
 - **Exemple** : `margin: 2rem;`

D- COMPARAISON ET CHOIX D'UNE UNITE

- **px** est fixe et ne change pas en fonction de la taille de la police ou de la taille de l'écran, ce qui le rend prévisible mais moins flexible.
- **%** est très flexible et s'adapte à l'environnement, mais peut parfois conduire à des résultats inattendus si l'élément parent change de taille.
- **em** est dynamique et s'adapte à la taille de la police de l'élément parent, ce qui est utile pour les composants qui doivent s'adapter à leur environnement.
- **rem** est également dynamique mais plus prévisible que **em**, car il est toujours relatif à la taille de la police de l'élément racine.

Le choix de l'unité dépend du contexte et des besoins spécifiques en matière de design. Les pixels sont parfaits pour la précision, les pourcentages pour la réactivité, et **em** et **rem** pour la flexibilité et l'adaptabilité, en particulier en ce qui concerne la taille de la police et l'accessibilité.

E - EXEMPLE PRATIQUE DE L'UTILISATION DE WIDTH ET HEIGHT

```

1 .container {
2   width: 80%; /* Largeur relative au conteneur parent */
3   margin: auto; /* Centrer le conteneur */
4 }
5
6 .element {
7   width: 50%; /* 50% de la largeur du conteneur */
8   height: 200px; /* Hauteur fixe */
9   box-sizing: border-box; /* Inclut padding et bordure dans la largeur/hauteur */
10  padding: 10px;
11  border: 1px solid black;
12 }

```

Dans cet exemple, **.container** est un conteneur qui prend 80% de la largeur de son parent, et **.element** est un élément à l'intérieur de **.container** qui prend 50% de la largeur de **.container**. La propriété **box-sizing: border-box;** assure que le padding et la bordure de **.element** sont inclus dans sa largeur et hauteur totales.

3) LES ESPACEMENTS (PADDING ET MARGIN)

La gestion des espacements autour et à l'intérieur d'un bloc en HTML et CSS est cruciale pour la mise en page et le design. Les propriétés **padding** et **margin** sont utilisées pour contrôler ces espacements. Elles font partie intégrante du modèle de boîte (box model) en CSS, qui comprend également la largeur (**width**), la hauteur (**height**), et la bordure (**border**).

A - PADDING (MARGIN INTERNE)

1. Définition :

- Le **padding** est l'espace entre le contenu d'un élément (comme le texte ou une image) et sa bordure.
- Il augmente la zone intérieure de l'élément, mais ne change pas sa taille externe (sauf si **box-sizing: border-box;** est utilisé).

2. Syntaxe :

- **padding: 10px;** applique un padding uniforme de 10 pixels sur tous les côtés.
- **padding: 10px 15px;** applique 10 pixels de padding en haut et en bas, et 15 pixels à gauche et à droite.
- **padding: 10px 15px 20px 25px;** applique des paddings différents pour chaque côté (haut, droite, bas, gauche).

3. Impact sur la mise en page :

- Le padding augmente l'espace à l'intérieur de l'élément, ce qui peut affecter la façon dont le contenu est affiché et interagit avec d'autres éléments.

B - MARGIN (MARGE EXTERNE)

1. Définition :

- La **margin** est l'espace entre la bordure d'un élément et les éléments adjacents.
- Contrairement au padding, la margin affecte l'espace externe autour de l'élément.

2. Syntaxe :

- **margin: 10px;** applique une margin uniforme de 10 pixels autour de l'élément.
- **margin: 10px 15px;** applique 10 pixels de margin en haut et en bas, et 15 pixels à gauche et à droite.
- **margin: 10px 15px 20px 25px;** applique des margins différentes pour chaque côté.

3. Marges collapsing :

- Dans certains cas, les marges verticales (haut et bas) de deux éléments adjacents peuvent se "collapser" ou se combiner en une seule marge. C'est un comportement unique aux marges verticales.

C - BOX-SIZING

- La propriété **box-sizing** détermine si le padding et la bordure sont inclus dans la largeur/hauteur totale de l'élément.
- **box-sizing: content-box;** (valeur par défaut) signifie que le padding et la bordure sont ajoutés à la largeur/hauteur définie.
- **box-sizing: border-box;** fait que la largeur/hauteur inclut le contenu, le padding et la bordure, mais pas la marge.

D - EXEMPLE PRATIQUE DE CES TROIS PROPRIETES

```

1 .box {
2   width: 300px;
3   padding: 20px; /* Espace à l'intérieur de la boîte */
4   margin: 30px; /* Espace autour de la boîte */
5   border: 5px solid black; /* Bordure de la boîte */
6   box-sizing: border-box; /* Inclut padding et bordure dans la largeur/hauteur */
7 }
```

Dans cet exemple, **.box** a un padding de 20px à l'intérieur, augmentant l'espace autour de son contenu. La margin de 30px crée un espace entre **.box** et les éléments environnants. La propriété **box-sizing: border-box;** assure que la largeur totale de **.box** reste à 300px, incluant le contenu, le padding et la bordure.

4) LA GESTION DU DEPASSEMENT AVEC OVERFLOW

La propriété CSS **overflow** est utilisée pour spécifier le comportement d'affichage du contenu d'un élément lorsque son contenu dépasse les dimensions définies pour cet élément. En d'autres termes, elle détermine comment gérer le contenu qui est trop grand pour tenir dans la boîte assignée (définie par les propriétés **width** et **height**).

A - VALEURS DE LA PROPRIETE OVERFLOW

1. **visible** (valeur par défaut) :
 - Le contenu déborde de la boîte de l'élément et est visible.
 - Exemple : **overflow: visible;**
2. **hidden** :
 - Le contenu qui dépasse est coupé et non visible.
 - Aucune barre de défilement n'est fournie pour voir le contenu restant.
 - Exemple : **overflow: hidden;**
3. **scroll** :
 - Des barres de défilement sont toujours présentes, que le contenu déborde ou non.
 - Permet à l'utilisateur de faire défiler le contenu pour voir ce qui est coupé.
 - Exemple : **overflow: scroll;**
4. **auto** :
 - Comportement similaire à **scroll**, mais les barres de défilement apparaissent uniquement si nécessaire (c'est-à-dire si le contenu déborde).

- Exemple : `overflow: auto;`

On peut aussi utiliser `overflow-x` ou `overflow-y` :

- Ces propriétés permettent de contrôler séparément le débordement sur l'axe horizontal (**x**) et vertical (**y**).
- Exemple : `overflow-x: auto; overflow-y: hidden;`
- Cela peut être utile pour créer des mises en page où, par exemple, vous voulez une barre de défilement horizontale mais pas verticale.

B - USAGES ET APPLICATIONS

1. Contrôle des débordements de contenu :

- `overflow` est souvent utilisé dans les interfaces où l'espace est limité, comme les menus déroulants, les panneaux latéraux, ou les widgets.

2. Crédit de zones de défilement :

- Pour les contenus longs ou larges, comme les tableaux ou les images, `overflow` peut être utilisé pour les contenir dans une zone définie avec défilement.

3. Design et mise en page :

- Dans certains designs, `overflow: hidden;` est utilisé pour masquer délibérément certaines parties du contenu pour créer des effets visuels spécifiques.

4. Amélioration de l'expérience utilisateur :

- Utiliser `overflow: auto;` peut améliorer l'expérience utilisateur en ne montrant les barres de défilement que lorsque c'est nécessaire.

C - EXEMPLE PRATIQUE

```

1 .box {
2   width: 200px;
3   height: 100px;
4   overflow: auto; /* Affiche une barre de défilement si le contenu déborde */
5   border: 1px solid black;
6 }
```

Dans cet exemple, `.box` a une largeur et une hauteur fixes. Si le contenu à l'intérieur de `.box` dépasse ces dimensions, une barre de défilement apparaîtra automatiquement pour permettre à l'utilisateur de voir le contenu restant.

5) BORDURE ET ARRONDI

Les propriétés CSS `border` et `border-radius` sont utilisées pour styliser les bordures des éléments HTML. Elles permettent de contrôler l'apparence des bordures d'un élément, y compris leur couleur, leur style, leur épaisseur, et leur arrondi.

A - PROPRIÉTÉ BORDER

La propriété `border` est une propriété raccourcie qui permet de définir l'épaisseur, le style et la couleur de la bordure d'un élément. Elle peut être décomposée en trois propriétés distinctes : `border-width`, `border-style`, et `border-color`.

1. Syntaxe :

- `border: [border-width] [border-style] [border-color];`
- Exemple : `border: 2px solid black;`

2. **border-width :**

- Définit l'épaisseur de la bordure. Peut être spécifiée en pixels, em, rem, etc.
- Exemple : **border-width: 4px;**

3. **border-style :**

- Définit le style de la bordure (solid, dashed, dotted, double, groove, ridge, inset, outset, none, hidden).
- Exemple : **border-style: dashed;**

4. **border-color :**

- Définit la couleur de la bordure. Peut accepter n'importe quelle valeur de couleur valide en CSS (nom de couleur, hexadécimal, rgb, rgba, hsl, hsla).
- Exemple : **border-color: #ff0000;**

B - PROPRIETE BORDER-RADIUS

La propriété **border-radius** est utilisée pour ajouter des coins arrondis à un élément. Elle peut prendre une ou plusieurs valeurs qui définissent le rayon de l'arrondi pour chaque coin.

1. Syntaxe :

- **border-radius: [value1] [value2] [value3] [value4];**
- Exemple : **border-radius: 10px;** (arrondit tous les coins avec un rayon de 10px)
- Vous pouvez spécifier un rayon différent pour chaque coin : **border-radius: 10px 20px 30px 40px;** (haut gauche, haut droit, bas droit, bas gauche)

2. Formes Elliptiques :

- **border-radius** peut également créer des formes elliptiques en utilisant deux valeurs séparées par une barre oblique (/). La première valeur définit le rayon horizontal, et la seconde le rayon vertical.
- Exemple : **border-radius: 50px 25px;** (rayon horizontal de 50px et vertical de 25px)

Il est possible de fixer la bordure d'un seul coin, par exemple **border-top-left-radius: 1rem;** pour le coin supérieur gauche.

D - EXEMPLE PRATIQUE

```

1 .box {
2   width: 200px;
3   height: 100px;
4   border: 3px solid blue; /* Bordure bleue solide de 3px */
5   border-radius: 15px; /* Coins arrondis avec un rayon de 15px */
6 }
```

Dans cet exemple, **.box** a une bordure bleue solide de 3px d'épaisseur et des coins arrondis avec un rayon de 15px. La propriété **border-radius** ajoute une touche esthétique en adoucissant les coins, ce qui peut être particulièrement utile pour des éléments comme des boutons, des cartes de contenu, ou des images.

6) LES IMAGES DE FOND

La propriété CSS **background-image** permet de définir une ou plusieurs images d'arrière-plan pour un élément HTML. Cette propriété est largement utilisée pour ajouter des visuels à l'arrière-plan des éléments, tels que des conteneurs, des en-têtes, des pieds de page, etc.

A - SYNTAXE DE BASE

```
element {
    background-image: url('chemin/vers/image.jpg');
}
```

- **url('chemin/vers/image.jpg')** : Spécifie le chemin de l'image que vous souhaitez utiliser comme arrière-plan. Le chemin peut être relatif ou absolu.

B - UTILISATION DE PLUSIEURS IMAGES

Vous pouvez également spécifier plusieurs images d'arrière-plan, séparées par des virgules :

```
element {
    background-image: url('image1.jpg'), url('image2.jpg');
}
```

Dans ce cas, **image1.jpg** sera superposée sur **image2.jpg**.

C - CONTROLE DE LA REPETITION ET DU POSITIONNEMENT

- **background-repeat** : Contrôle si et comment l'image d'arrière-plan se répète. Les valeurs courantes incluent **repeat**, **repeat-x**, **repeat-y**, et **no-repeat**.
- **background-position** : Définit la position de l'image d'arrière-plan. Les valeurs peuvent être en pourcentages, en pixels, ou des mots-clés comme **center**, **top**, **bottom**, **left**, **right**.

D - TAILLE DE L'IMAGE D'ARRIERE-PLAN

- **background-size** : Permet de spécifier la taille de l'image d'arrière-plan. Les valeurs courantes incluent **cover** (l'image est redimensionnée pour couvrir entièrement l'élément tout en conservant ses proportions), **contain** (l'image est redimensionnée pour s'adapter entièrement à l'intérieur de l'élément), ou des dimensions spécifiques (comme **100px 200px**).

```
1  div {
2      width: 300px;
3      height: 200px;
4      background-image: url('background.jpg');
5      background-repeat: no-repeat;
6      background-position: center;
7      background-size: cover;
8  }
```

Dans cet exemple, un **div** est stylisé avec une image d'arrière-plan. L'image ne se répète pas (**no-repeat**), est centrée (**background-position: center**), et couvre toute la zone du **div** (**background-size: cover**), en s'ajustant au besoin pour couvrir l'espace sans déformer l'image.

E - APPLICATIONS

La propriété **background-image** est utilisée dans divers contextes, comme pour :

- Ajouter des images décoratives à des sections de pages web.
- Créer des en-têtes ou des bannières avec des images d'arrière-plan.
- Styliser des éléments interactifs, comme des boutons ou des liens, avec des images d'arrière-plan qui changent au survol.

En combinant **background-image** avec d'autres propriétés d'arrière-plan, vous pouvez créer des designs complexes et visuellement attrayants pour vos pages web.

7) UN FORMULAIRE DE BASE

La création d'un formulaire de base en HTML implique l'utilisation de la balise **<form>** qui est le conteneur et au minimum un champs de saisie et un élément de validation. Ici, nous utiliserons une **<input>** de type **text**, et un **<input>** de type **submit**. Voici comment ces éléments s'articulent ensemble pour former un formulaire fonctionnel.

A - STRUCTURE DE BASE D'UN FORMULAIRE

1. Balise **<form>** :

- La balise **<form>** sert de conteneur pour les éléments du formulaire.
- Elle peut inclure des attributs comme **action** (l'URL où les données du formulaire seront envoyées) et **method** (la méthode HTTP pour l'envoi des données, généralement **GET** ou **POST**).

```
<form action="url_de_traitement" method="post">
    <!-- Éléments du formulaire ici -->
</form>
```

2. Champs de Saisie (**<input type="text">**) :

- Les champs de saisie pour le texte sont créés avec **<input type="text">**.
- Chaque champ peut avoir des attributs comme **name** (nom du champ, important pour le traitement des données), **id** (identifiant unique pour le champ), et **placeholder** (texte indicatif qui s'affiche dans le champ).

```
<input type="text" name="nom_utilisateur" id="nom_utilisateur" placeholder="Votre nom">
```

3. Bouton de Soumission (**<input type="submit">**) :

- Un bouton de soumission est créé avec **<input type="submit">**.
- Cet élément peut avoir un attribut **value** qui définit le texte affiché sur le bouton.

```
<input type="submit" value="Envoyer">
```

B - MISE EN ŒUVRE DANS UN EXEMPLE

```

1 <form action="traitement.php" method="post">
2   <label for="nom_utilisateur">Nom :</label>
3   <input type="text" name="nom_utilisateur" id="nom_utilisateur" placeholder="Votre nom">
4
5   <label for="email">Email :</label>
6   <input type="text" name="email" id="email" placeholder="Votre email">
7
8   <input type="submit" value="Envoyer">
9 </form>

```

Dans cet exemple :

- Le formulaire contient deux champs de saisie : un pour le nom de l'utilisateur et un autre pour l'email.
- Chaque champ de saisie est précédé d'une balise **<label>**, qui améliore l'accessibilité en associant le texte du label au champ correspondant.
- Le formulaire sera envoyé à **traitement.php** via la méthode POST lorsque l'utilisateur cliquera sur "Envoyer".

C - POINTS CLEFS

- **Labels** : Utiliser des **<label>** avec l'attribut **for** correspondant à l'**id** des champs de saisie améliore l'accessibilité et l'expérience utilisateur.
- **Traitement des Données** : Le fichier ou le script spécifié dans l'attribut **action** du formulaire (**traitement.php** dans cet exemple) est responsable de traiter les données soumises.
- **Sécurité** : Toujours valider et nettoyer les données côté serveur pour éviter les vulnérabilités de sécurité comme les injections SQL.

8) LA BALISE LABEL

La balise **<label>** en HTML est utilisée pour associer un texte descriptif à un élément de formulaire spécifique, comme une case à cocher, un bouton radio, ou un champ de saisie. L'utilisation de **<label>** améliore l'accessibilité et l'expérience utilisateur, car elle permet aux utilisateurs de cliquer sur le texte du label pour interagir avec l'élément de formulaire associé.

A - FONCTIONNALITES ESSENTIELLES

1. **Association avec un champ de formulaire :**
 - Un **<label>** est associé à un élément de formulaire soit en plaçant cet élément à l'intérieur de la balise **<label>**, soit en utilisant l'attribut **for** dans la balise **<label>**, qui doit correspondre à l'**id** de l'élément de formulaire.
2. **Amélioration de l'accessibilité :**
 - Les labels sont particulièrement utiles pour améliorer l'accessibilité, car ils fournissent un texte descriptif pour les lecteurs d'écran et d'autres technologies d'assistance.
3. **Facilité d'utilisation :**
 - Cliquer sur le texte du label déclenche l'élément de formulaire associé, ce qui est pratique pour des éléments plus petits comme les cases à cocher et les boutons radio.

B - DEUX EXEMPLES D'UTILISATION

- Exemple 1 : Label enveloppant l'élément

```

1  <label>
2    Nom :
3    <input type="text" name="nom_utilisateur">
4  </label>
```

Dans cet exemple, l'utilisateur peut cliquer sur "Nom :" pour placer le curseur dans le champ de saisie.

- Exemple 2 : Label utilisant l'attribut **for**

```

1  <label for="nom_utilisateur">Nom :</label>
2  <input type="text" id="nom_utilisateur" name="nom_utilisateur">
```

Ici, le **<label>** est associé à l'**<input>** par l'intermédiaire de l'attribut **for**, qui correspond à l'**id** de l'**<input>**. Cliquer sur "Nom :" positionnera le curseur dans le champ associé.

C - BONNES PRATIQUES

- **Utiliser des Labels pour Chaque Élément de Formulaire** : Chaque élément de formulaire devrait avoir un label associé pour une meilleure accessibilité.
- **Correspondance des for et id** : Lorsque vous utilisez l'attribut **for**, assurez-vous qu'il correspond exactement à l'**id** de l'élément de formulaire.
- **Texte Clair et Descriptif** : Le texte du label doit être clair et suffisamment descriptif pour que l'utilisateur comprenne ce que l'élément de formulaire représente.

En résumé, l'utilisation correcte de la balise **<label>** rend vos formulaires plus accessibles et plus faciles à utiliser, améliorant ainsi l'expérience globale de l'utilisateur sur votre site web.

9) LES DIFFERENTS TYPES D'INPUT

Les éléments **<input>** en HTML peuvent prendre divers types, chacun étant conçu pour une saisie de données spécifique. En dehors du type **text**, voici quelques-uns des types d'**input** les plus couramment utilisés :

1. **type="tel"** :
 - Utilisé pour les champs de saisie de numéros de téléphone.
 - Les navigateurs peuvent optimiser le clavier sur les appareils mobiles pour faciliter la saisie des numéros.
2. **type="email"** :
 - Conçu pour les adresses e-mail.
 - Les navigateurs peuvent valider automatiquement le format de l'adresse e-mail et optimiser le clavier sur les appareils mobiles.
3. **type="date"** :
 - Permet à l'utilisateur de sélectionner une date à l'aide d'un sélecteur de date.
 - Le format de la date sélectionnée est généralement **YYYY-MM-DD**.
4. **type="url"** :
 - Pour les adresses web (URLs).
 - Les navigateurs peuvent valider le format de l'URL.

5. **type="password"** :

- Utilisé pour les champs de mot de passe.
- Les caractères saisis sont masqués pour la confidentialité.

6. **type="number"** :

- Pour la saisie de nombres.
- Les navigateurs peuvent fournir des contrôles pour augmenter ou diminuer la valeur.

7. **type="range"** :

- Un curseur pour sélectionner une valeur dans une plage.
- Utile pour les réglages comme le volume, la luminosité, etc.

8. **type="checkbox"** :

- Pour les cases à cocher, permettant une sélection multiple.

9. **type="radio"** :

- Boutons radio pour une sélection unique parmi plusieurs choix.

10. **type="file"** :

- Permet aux utilisateurs de sélectionner un ou plusieurs fichiers à télécharger.

11. **type="color"** :

- Un sélecteur de couleur.

12. **type="hidden"** :

- Pour les données qui doivent être envoyées avec le formulaire mais ne doivent pas être visibles ou modifiables par l'utilisateur.

```

1  <input type="tel" name="telephone">
2
3  <input type="email" name="email">
4
5  <input type="date" name="date_naissance">
6
7  <input type="url" name="website">
8
9  <input type="password" name="password">
10
11 <input type="number" name="quantite">
12
13 <input type="range" name="volume">
14
15 <input type="checkbox" name="option1" value="Option 1">
16
17 <input type="radio" name="genre" value="masculin">
18
19 <input type="file" name="document">
20
21 <input type="color" name="couleur_favorite">
22
23 <input type="hidden" name="identifiant_cache" value="12345">
```

Chaque type d'**input** est conçu pour faciliter la saisie de données spécifiques, améliorant ainsi l'expérience utilisateur et la précision des données collectées.

10) TEXTAREA ET SELECT

Les balises `<textarea>` et `<select>` sont des éléments de formulaire HTML utilisés pour recueillir des entrées utilisateur, mais chacune a une fonction spécifique et une manière d'utilisation distincte.

A - BALISE `<TEXTAREA>`

La balise `<textarea>` est utilisée pour créer une zone de texte multi-lignes, permettant aux utilisateurs d'entrer du texte sur plusieurs lignes. Elle est souvent utilisée pour des champs de saisie plus importants, comme les commentaires ou les descriptions. Ses caractéristiques principales sont :

- **Multi-lignes** : Contrairement à `<input type="text">`, qui est limité à une seule ligne, `<textarea>` permet la saisie sur plusieurs lignes.
- **Attributs** : Comme `rows` et `cols` pour définir la hauteur et la largeur du `<textarea>`. D'autres attributs communs incluent `name`, `placeholder`, `readonly`, `disabled`, etc.
- **Contenu initial** : Tout texte placé entre les balises ouvrante et fermante de `<textarea>` sera affiché comme contenu initial.

```
1 <textarea name="message" rows="4" cols="50" placeholder="Entrez votre message ici">
2 </textarea>
```

Dans cet exemple, un `<textarea>` est créé avec un espace pour un message de quatre lignes et cinquante colonnes.

B - BALISE `<SELECT>`

La balise `<select>` crée une liste déroulante, permettant aux utilisateurs de choisir une option parmi plusieurs. Elle est souvent utilisée pour les formulaires où l'utilisateur doit sélectionner une option dans une liste prédéfinie. Ses caractéristiques principales sont :

- **Options** : Les options de la liste déroulante sont définies par des balises `<option>` à l'intérieur de `<select>`.
- **Attributs** : Comme `name` (pour identifier les données du formulaire) et `multiple` (permettant de sélectionner plusieurs options).
- **Balise `<option>`** : Chaque `<option>` peut avoir des attributs comme `value` (la valeur à envoyer lors de la soumission du formulaire) et `selected` (pour pré-sélectionner une option).

```
1 <select name="fruit">
2   <option value="apple">Pomme</option>
3   <option value="banana">Banane</option>
4   <option value="orange">Orange</option>
5 </select>
```

Dans cet exemple, un menu déroulant est créé avec trois options : pomme, banane et orange. L'utilisateur peut sélectionner l'une de ces options.

C - UTILISATIONS

- `<textarea>` : Idéal pour des champs de texte plus longs, comme des commentaires, des messages, ou des descriptions.

- **<select>** : Utilisé pour des choix parmi une liste d'options, comme des sélections de pays, de langues, ou de préférences.

Ces éléments sont essentiels pour créer des formulaires interactifs et conviviaux sur les pages web.

11) FIELDSET ET LEGEND

Les balises **<fieldset>** et **<legend>** en HTML sont utilisées ensemble pour organiser et étiqueter des groupes d'éléments au sein d'un formulaire. Elles aident à structurer le formulaire de manière logique et à améliorer son accessibilité et sa lisibilité.

A - BALISE <FIELDSET>

La balise **<fieldset>** est utilisée pour regrouper plusieurs contrôles de formulaire ainsi que leurs étiquettes (**<label>**) au sein d'une même section. Elle crée visuellement un cadre autour de son contenu, indiquant que les éléments qu'elle contient sont liés. Ses caractéristiques principales sont :

- **Regroupement** : Utilisée pour regrouper des éléments liés dans un formulaire, comme un ensemble de boutons radio ou de cases à cocher.
- **Accessibilité** : Améliore l'accessibilité en regroupant les éléments de formulaire connexes.

```

1  <fieldset>
2      <legend>Informations Personnelles</legend>
3      <label for="nom">Nom :</label>
4      <input type="text" id="nom" name="nom">
5      <label for="email">Email :</label>
6      <input type="email" id="email" name="email">
7  </fieldset>
8

```

Dans cet exemple, un **<fieldset>** est utilisé pour regrouper des champs de saisie pour le nom et l'email.

B - BALISE <LEGEND>

La balise **<legend>** est utilisée en conjonction avec **<fieldset>** pour fournir une étiquette ou un titre au groupe d'éléments de formulaire. Elle est placée à l'intérieur de **<fieldset>** et s'affiche généralement en haut du cadre créé par **<fieldset>**. Ses caractéristiques principales sont :

- **Titre du groupe** : Fournit un titre ou une étiquette pour le groupe d'éléments de formulaire dans le **<fieldset>**.
- **Visibilité** : Améliore la lisibilité et aide les utilisateurs à comprendre rapidement le contexte du groupe d'éléments.

```

1  <fieldset>
2      <legend>Choix de la Langue</legend>
3      <label><input type="radio" name="langue" value="français"> Français</label>
4      <label><input type="radio" name="langue" value="anglais"> Anglais</label>
5  </fieldset>

```

Dans cet exemple, **<legend>** est utilisé pour donner un titre au groupe de boutons radio qui permettent de choisir une langue.

C - UTILISATIONS

- **Organisation des formulaires :** `<fieldset>` et `<legend>` sont particulièrement utiles dans les formulaires longs et complexes pour les diviser en sections plus gérables.
- **Amélioration de l'expérience utilisateur :** Ils aident à guider l'utilisateur à travers les différentes parties d'un formulaire, en rendant clair quelles sont les informations demandées dans chaque section.

En résumé, l'utilisation de `<fieldset>` et `<legend>` contribue à la création de formulaires bien structurés, faciles à naviguer et accessibles, améliorant ainsi l'expérience globale de l'utilisateur.

12) DECOUVERTE DES TABLEAUX HTML - TABLE, TR ET TD

Les balises `<table>`, `<tr>`, et `<td>` en HTML sont utilisées pour créer et structurer des tableaux de données. Chacune de ces balises joue un rôle spécifique dans la construction et l'organisation du tableau.

A - BALISE `<TABLE>`

La balise `<table>` est le conteneur principal pour les éléments d'un tableau. Elle définit l'espace dans lequel le tableau sera affiché. Ses caractéristiques principales sont :

- **Conteneur de tableau :** Englobe tous les autres éléments liés au tableau, comme les lignes, les cellules, les en-têtes, etc.
- **Structure de base :** Sert de base pour la construction du tableau.

```
<table>
  <!-- Lignes (tr) et cellules (td) du tableau ici -->
</table>
```

B - BALISE `<TR>`

La balise `<tr>` (table row) est utilisée pour créer une ligne dans le tableau. Chaque `<tr>` définit une nouvelle ligne horizontale dans le tableau. Ses caractéristiques principales sont :

- **Création de lignes :** Utilisée pour définir les lignes horizontales du tableau.
- **Conteneur pour les cellules :** Contient des balises `<td>` (cellules de données) ou `<th>` (cellules d'en-tête).

```
<tr>
  <!-- Cellules (td) ou en-têtes (th) de cette ligne ici -->
</tr>
```

C - BALISE `<TD>`

La balise `<td>` (table data) représente une cellule de données dans le tableau, habituellement utilisée pour le contenu du tableau. Ses caractéristiques principales sont :

- **Cellules de données :** Utilisée pour insérer des données dans les lignes du tableau.
- **Appartient à `<tr>` :** Doit toujours être placée à l'intérieur d'une balise `<tr>`.

```
<td>Contenu de la cellule</td>
```

D - EXEMPLE UTILISANT CES TROIS BALISES

```

1  <table>
2      <tr>
3          <td>Cellule 1, Ligne 1</td>
4          <td>Cellule 2, Ligne 1</td>
5      </tr>
6      <tr>
7          <td>Cellule 1, Ligne 2</td>
8          <td>Cellule 2, Ligne 2</td>
9      </tr>
10     </table>

```

Dans cet exemple, un tableau avec deux lignes et deux colonnes est créé. Chaque ligne (`<tr>`) contient deux cellules (`<td>`), chacune avec son propre contenu.

E - UTILISATIONS

- **Présentation de données** : Les tableaux sont idéaux pour présenter des données sous forme de grilles, comme des horaires, des listes de prix, ou des spécifications.
- **Organisation de contenu** : Ils peuvent également être utilisés pour structurer le contenu sur une page web, bien que cette pratique soit moins courante avec l'avènement des CSS pour la mise en page.

Il est important de noter que l'utilisation de tableaux pour la mise en page de sites web entiers est déconseillée en faveur des techniques de mise en page CSS plus modernes et flexibles. Les tableaux doivent être principalement utilisés pour présenter des données tabulaires.

13) STRUCTURATION DE TABLEAU - THEAD, TBODY ET TFOOT

La structuration d'un tableau HTML peut être améliorée en utilisant des balises supplémentaires telles que `<th>`, `<thead>`, `<tbody>`, `<tfoot>`, et `<caption>`. Ces éléments permettent de créer des tableaux plus organisés et accessibles, avec une distinction claire entre les en-têtes, le corps principal, le pied de tableau, et un titre descriptif.

A - BALISE `<TH>`

`<th>` (table header) est utilisée pour définir une cellule d'en-tête dans un tableau. Les cellules `<th>` sont généralement en gras et centrées par défaut, indiquant qu'elles contiennent des informations d'en-tête pour les colonnes ou les lignes.

`<th>Nom</th>`

B - BALISE <THEAD>

<**thead**> (table head) est utilisée pour grouper l'ensemble des lignes d'en-tête d'un tableau. Elle aide à séparer l'en-tête du reste du tableau, ce qui est utile pour le style et peut également aider les technologies d'assistance à comprendre la structure du tableau.

```
1  <thead>
2    <tr>
3      <th>Nom</th>
4      <th>Age</th>
5    </tr>
6  </thead>
```

C - BALISE <TBODY>

<**tbody**> (table body) est utilisée pour grouper le corps principal du tableau, contenant toutes les données à l'exception des en-têtes et des pieds de tableau.

```
1  <tbody>
2    <tr>
3      <td>Jean</td>
4      <td>30</td>
5    </tr>
6    <!-- Plus de lignes de données ici -->
7  </tbody>
```

D - BALISE <TFOOT>

<**tfoot**> (table footer) est utilisée pour grouper les lignes de pied de tableau. Les informations placées ici sont généralement des résumés ou des totaux des données contenues dans le tableau.

```
1  <tfoot>
2    <tr>
3      <td>Total</td>
4      <td>100</td>
5    </tr>
6  </tfoot>
```

E - BALISE <CAPTION>

<**caption**> fournit un titre ou une légende pour le tableau. Il est généralement placé immédiatement après la balise d'ouverture <**table**>.

<**caption**>Liste des Participants</**caption**>

F - EXEMPLE COMPLET

```

1  <table>
2      <caption>Liste des Participants</caption>
3      <thead>
4          <tr>
5              <th>Nom</th>
6              <th>Age</th>
7          </tr>
8      </thead>
9      <tbody>
10     <tr>
11         <td>Jean</td>
12         <td>30</td>
13     </tr>
14     <!-- Plus de lignes de données ici -->
15 </tbody>
16 <tfoot>
17     <tr>
18         <td>Total</td>
19         <td>100</td>
20     </tr>
21 </tfoot>
22 </table>

```

Dans cet exemple, le tableau est clairement structuré avec un en-tête (<**thead**>), un corps (<**tbody**>), un pied de tableau (<**tfoot**>), et une légende (<**caption**>). Cette structure aide à la fois à la présentation visuelle et à l'accessibilité du tableau.

14) BORDURE DE TABLEAU - BORDER ET BORDER-COLLAPSE

La gestion des bordures dans les tableaux HTML peut être effectuée à l'aide des propriétés CSS **border** et **border-collapse**. Ces propriétés permettent de contrôler l'apparence des bordures des tableaux et de leurs cellules.

A - PROPRIETE BORDER

La propriété **border** en CSS est utilisée pour définir le style, la largeur et la couleur des bordures d'un élément. Dans le contexte des tableaux, elle peut être appliquée à la balise **<table>** elle-même, ainsi qu'aux balises **<th>** et **<td>** pour les cellules individuelles.

```
table, th, td {
    border: 1px solid black;
}
```

Dans cet exemple, toutes les cellules du tableau (**<th>** et **<td>**) ainsi que le tableau lui-même (**<table>**) auront des bordures solides de 1 pixel de largeur de couleur noire.

B - PROPRIETE BORDER-COLLAPSE

La propriété **border-collapse** est spécifique aux tableaux et détermine comment les bordures des cellules sont fusionnées ou séparées.

- **border-collapse: separate;** (valeur par défaut) : Les bordures de chaque cellule sont affichées séparément. Avec cette valeur, vous pouvez également utiliser la propriété **border-spacing** pour contrôler l'espace entre les bordures des cellules.
- **border-collapse: collapse;** : Les bordures adjacentes des cellules sont fusionnées en une seule bordure. C'est souvent plus esthétique et est généralement préféré pour un design moderne.

```
table {
    border-collapse: collapse;
}
```

Avec **border-collapse: collapse;**, les bordures des cellules adjacentes du tableau seront fusionnées, donnant une apparence plus unifiée et épurée.

C - EXEMPLE METTANT EN ŒUVRE CES DEUX PROPRIETES

```
1  table, th, td {
2      border: 1px solid black;
3  }
4
5  table {
6      border-collapse: collapse;
7  }
```

Dans cet exemple complet, le tableau aura des bordures noires solides autour de chaque cellule, et ces bordures seront fusionnées pour un aspect plus net et plus cohérent. Cela améliore l'esthétique du tableau et rend les données plus faciles à lire et à comprendre.

15) FUSION DE CELLULES - COLSPAN ET ROWSPAN

La fusion de cellules dans les tableaux HTML est réalisée à l'aide des attributs **colspan** et **. Ces attributs permettent de fusionner plusieurs cellules horizontalement ou verticalement, respectivement, créant ainsi des cellules plus grandes qui s'étendent sur plusieurs colonnes ou lignes.**

A - ATTRIBUT COLSPAN

L'attribut **colspan** est utilisé pour fusionner des cellules horizontalement sur plusieurs colonnes.

- Usage** : Il est ajouté à une cellule de tableau (**<td>** ou **<th>**) pour indiquer combien de colonnes cette cellule doit englober.
- Valeur** : La valeur de **colspan** est un nombre entier indiquant le nombre total de colonnes que la cellule doit couvrir.

```

1  <table>
2    <tr>
3      <td colspan="2">Cette cellule s'étend sur deux colonnes</td>
4    </tr>
5    <tr>
6      <td>Cellule normale</td>
7      <td>Une autre cellule</td>
8    </tr>
9  </table>

```

Dans cet exemple, la première cellule de la première ligne s'étend sur deux colonnes.

B - ATTRIBUT ROWSPAN

L'attribut **est utilisé pour fusionner des cellules verticalement sur plusieurs lignes.**

- Usage** : Il est ajouté à une cellule de tableau pour indiquer combien de lignes cette cellule doit englober.
- Valeur** : La valeur de **est un nombre entier indiquant le nombre total de lignes que la cellule doit couvrir.**

```

1  <table>
2    <tr>
3      <td rowspan="2">Cette cellule s'étend sur deux lignes</td>
4      <td>Cellule 1</td>
5    </tr>
6    <tr>
7      <td>Cellule 2</td>
8    </tr>
9  </table>

```

Dans cet exemple, la première cellule de la première ligne s'étend sur deux lignes.

C - COMBINAISON DE COLSPAN ET ROWSPAN

Il est également possible de combiner **colspan** et **pour créer une cellule qui s'étend à la fois horizontalement et verticalement sur plusieurs colonnes et lignes.**

```

1 <table>
2   <tr>
3     <td rowspan="2" colspan="2">Cette cellule s'étend sur deux lignes et deux colonnes</td>
4     <td>Cellule 1</td>
5   </tr>
6   <tr>
7     <td>Cellule 2</td>
8   </tr>
9 </table>
10

```

Dans cet exemple, la première cellule s'étend sur deux colonnes et deux lignes, créant une grande cellule dans le coin supérieur gauche du tableau.

D - POINTS ESSENTIELS

- Planification** : Lors de l'utilisation de **colspan** et **rowspan**, il est important de planifier soigneusement la structure du tableau pour éviter les erreurs de disposition.
- Accessibilité** : Bien que ces attributs puissent améliorer la présentation des données, il est essentiel de s'assurer que le tableau reste accessible et lisible, en particulier pour les utilisateurs de lecteurs d'écran.

En résumé, **colspan** et **rowspan** sont des outils puissants pour personnaliser la disposition des tableaux HTML, permettant une présentation plus flexible et détaillée des données.

16) LES OMBRAGES AVEC BOX-SHADOW

La propriété CSS **box-shadow** est utilisée pour ajouter des ombres aux éléments HTML, créant un effet de profondeur qui peut améliorer l'esthétique de la conception d'une page web. Cette propriété peut être appliquée à presque tous les éléments pour ajouter une ombre autour de leur boîte de bordure.

A - SYNTAXE DE BOX-SHADOW

La propriété **box-shadow** peut prendre plusieurs valeurs pour définir l'apparence de l'ombre :

- Décalage horizontal (x-offset)** : La distance horizontale à laquelle l'ombre est décalée par rapport à l'élément. Une valeur positive décale l'ombre vers la droite, tandis qu'une valeur négative la décale vers la gauche.
- Décalage vertical (y-offset)** : La distance verticale à laquelle l'ombre est décalée. Une valeur positive décale l'ombre vers le bas, tandis qu'une valeur négative la décale vers le haut.
- Flou (blur-radius) (optionnel)** : Le rayon de flou de l'ombre. Plus la valeur est élevée, plus l'ombre est floue et plus grande. Si non spécifié ou si la valeur est **0**, l'ombre sera nette.
- Étalement (spread-radius) (optionnel)** : La taille de l'ombre. Des valeurs positives étendent l'ombre, des valeurs négatives la réduisent.
- Couleur** : La couleur de l'ombre. Peut être spécifiée avec n'importe quel format de couleur valide en CSS (nom, hexadécimal, rgb, rgba, etc.).
- Inset (optionnel)** : Si **inset** est spécifié, l'ombre est appliquée à l'intérieur de l'élément plutôt qu'à l'extérieur.

```

1 .box {
2   width: 100px;
3   height: 100px;
4   background-color: blue;
5   box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.75);
6 }

```

Dans cet exemple, un élément avec la classe `.box` aura une ombre décalée de 10 pixels vers la droite et 10 pixels vers le bas, avec un flou de 5 pixels et une couleur noire semi-transparente.

B - OMBRE INTERNE

Pour créer une ombre interne (à l'intérieur de l'élément), utilisez le mot-clé `inset` :

```

1 .box-inner {
2   box-shadow: inset 5px 5px 10px rgba(0, 0, 0, 0.5);
3 }

```

Ici, l'ombre sera à l'intérieur de l'élément, créant un effet de creux.

POINTS CLES

- Plusieurs ombres** : Vous pouvez appliquer plusieurs ombres à un élément en séparant chaque ombre par une virgule.
- Performance** : Bien que les ombres puissent améliorer l'esthétique, elles peuvent aussi affecter les performances, en particulier lorsqu'elles sont utilisées sur de grands éléments ou en grande quantité.
- Compatibilité** : `box-shadow` est bien pris en charge dans les navigateurs modernes, mais il est toujours bon de vérifier la compatibilité pour les projets nécessitant un support de navigateurs plus anciens.

En résumé, `box-shadow` est un outil puissant pour ajouter de la profondeur et de l'intérêt visuel aux éléments de votre page web.

17) LES COLONNES

La gestion des colonnes en CSS est un moyen efficace de présenter du contenu textuel, comme des articles de journal ou des magazines, en le divisant en colonnes plus gérables et esthétiquement agréables. Les propriétés CSS telles que `column-count`, `column-gap`, `column-rule-width`, `column-rule-color`, et `column-rule-style` permettent de contrôler finement l'apparence et le comportement de ces colonnes.

A - PROPRIETE COLUMN-COUNT

- Usage** : Définit le nombre de colonnes dans lesquelles le contenu de l'élément doit être divisé.
- Valeur** : Un nombre entier indiquant le nombre de colonnes.

```

.container {
  column-count: 3;
}

```

Dans cet exemple, le contenu de `.container` sera divisé en trois colonnes.

B - PROPRIETE COLUMN-GAP

- **Usage** : Spécifie l'espace entre les colonnes.
- **Valeur** : La taille de l'écart, qui peut être en unités (comme px, em, etc.) ou en pourcentage.

```
.container {
  column-gap: 20px;
}
```

Ici, il y aura un espace de 20 pixels entre chaque colonne.

C - PROPRIETES COLUMN-RULE-*

Ces propriétés sont utilisées pour définir une règle (ligne verticale) entre les colonnes. Elles sont similaires à la propriété `border`, mais s'appliquent aux espaces entre les colonnes.

- **column-rule-width** : Définit l'épaisseur de la règle entre les colonnes.

```
.container {
  column-rule-width: 2px;
}
```

- **column-rule-color** : Définit la couleur de la règle entre les colonnes.

```
.container {
  column-rule-color: blue;
}
```

- **column-rule-style** : Définit le style de la règle (solide, pointillé, en pointillés, etc.).

```
.container {
  column-rule-style: dashed;
}
```

D - EXEMPLE COMPLET

```
1 .container {
2   column-count: 3;
3   column-gap: 20px;
4   column-rule-width: 1px;
5   column-rule-color: grey;
6   column-rule-style: solid;
7 }
```

Dans cet exemple complet, le contenu de `.container` sera divisé en trois colonnes, avec un écart de 20 pixels entre chaque colonne et une règle solide grise de 1 pixel d'épaisseur entre elles.

E - POINTS CLEFS

- **Responsive design** : Lors de la conception de mises en page réactives, il est important de considérer comment les colonnes se comporteront sur différents appareils. Parfois, réduire le nombre de colonnes sur des écrans plus petits peut améliorer la lisibilité.
- **Compatibilité navigateur** : Bien que largement supportées, les propriétés de colonnes CSS peuvent se comporter légèrement différemment dans différents navigateurs. Il est conseillé de tester votre mise en page sur plusieurs navigateurs pour assurer la cohérence.

En utilisant ces propriétés CSS, vous pouvez créer des mises en page de texte élaborées et lisibles, similaires à celles que l'on trouve dans les magazines et les journaux.

18) LE SURVOL AVEC :HOVER

La pseudo-classe **:hover** en CSS est utilisée pour appliquer un style à un élément lorsqu'il est survolé par le curseur de la souris. C'est un moyen efficace d'améliorer l'interactivité et la réactivité visuelle d'une page web, en fournissant un retour visuel immédiat lorsque les utilisateurs passent leur souris sur des éléments spécifiques.

A - UTILISATION DE :HOVER

La pseudo-classe **:hover** peut être appliquée à presque tous les éléments HTML. Elle est souvent utilisée pour modifier l'apparence des liens, des boutons, des images, et d'autres éléments interactifs pour indiquer qu'ils peuvent être cliqués ou interagis.

```
a:hover {
  color: red;
}
```

Dans cet exemple, tous les liens ([<a>](#)) sur la page changeront de couleur et deviendront rouges lorsqu'ils seront survolés par la souris.

APPLICATIONS COURANTES DE :HOVER

1. **Changement de couleur** : Modifier la couleur de fond ou de texte d'un bouton ou d'un lien lors du survol pour indiquer qu'il est interactif.
2. **Effets de transition** : Utiliser **:hover** en combinaison avec la propriété **transition** pour créer des animations fluides lors du survol.
3. **Modification de l'opacité** : Rendre un élément plus transparent ou plus opaque lors du survol.
4. **Décoration de texte** : Ajouter ou enlever des décorations de texte, comme **text-decoration: underline;**, sur des éléments textuels.

```

1 button:hover {
2   background-color: blue;
3   color: white;
4 }
5
6 .card:hover {
7   transform: scale(1.05);
8   transition: transform 0.3s;
9 }
10
11 img:hover {
12   opacity: 0.8;
13 }
14
15 .text:hover {
16   text-decoration: underline;
17 }

```

B - POINTS ESSENTIELS

- Accessibilité** : Bien que `:hover` soit un excellent outil pour améliorer l'interactivité, il est important de s'assurer que les changements de style ne nuisent pas à la lisibilité ou à l'accessibilité. Par exemple, évitez de changer la taille du texte de manière à ce qu'il devienne difficile à lire.
- Compatibilité mobile** : Sur les appareils tactiles sans curseur, `:hover` n'a pas le même effet. Il est donc important de s'assurer que votre site reste utilisable et esthétique même sans les effets de survol.
- Subtilité** : Les effets de survol doivent être subtils et ne pas détourner excessivement l'attention de l'utilisateur de l'expérience globale du site.

En résumé, `:hover` est un outil puissant pour améliorer l'expérience utilisateur en fournissant des indications visuelles et interactives, rendant les interfaces web plus intuitives et engageantes.

19) LE MASQUAGE (DISPLAY:NONE)

La déclaration CSS `display: none;` est utilisée pour masquer complètement un élément de la page web. Lorsqu'elle est appliquée à un élément, celui-ci est retiré du flux de la page, ce qui signifie qu'il n'occupe plus d'espace et qu'il est totalement invisible à l'utilisateur. C'est un moyen efficace de retirer des éléments de l'interface sans les supprimer du code HTML.

A - FONCTIONNEMENT DE DISPLAY: NONE;

Lorsque vous appliquez `display: none;` à un élément, voici ce qui se passe :

- Invisibilité** : L'élément devient complètement invisible à l'utilisateur.
- Retrait du flux** : L'élément est retiré du flux normal du document. Cela signifie qu'il n'occupe plus d'espace dans la mise en page. Les éléments autour de lui se comporteront comme s'il n'existe pas.
- Non interactif** : L'élément n'est pas seulement invisible, mais il est également retiré de l'arborescence de rendu, ce qui signifie qu'il ne peut pas être interactif (il ne peut pas recevoir de clics, par exemple).

```
.hidden {
  display: none;
}
```

Dans cet exemple, tous les éléments avec la classe **hidden** seront complètement masqués de la page.

B - COMPARAISON AVEC **VISIBILITY: HIDDEN;**

Il est important de distinguer **display: none;** de **visibility: hidden;**. Lorsque **visibility: hidden;** est appliqué à un élément, celui-ci reste dans le flux du document (occupant de l'espace) mais devient invisible. En revanche, **display: none;** retire l'élément du flux, libérant l'espace qu'il occupait.

C - UTILISATIONS COURANTES

- **Masquer Dynamiquement des éléments** : Utilisé en combinaison avec JavaScript pour masquer et afficher des éléments en réponse à des actions de l'utilisateur.
- **Améliorer l'accessibilité** : Masquer des éléments qui ne doivent pas être affichés à certains utilisateurs, tout en les gardant accessibles pour les lecteurs d'écran lorsque cela est approprié.
- **Gestion de l'espace de la page** : Utilisé pour créer des mises en page réactives où certains éléments ne doivent apparaître que dans certaines conditions ou sur certains appareils.

D - POINTS CLES

- **SEO et accessibilité** : Bien que **display: none;** soit utile, son utilisation excessive peut avoir des implications en termes de SEO et d'accessibilité. Les moteurs de recherche peuvent pénaliser les sites qui masquent un grand contenu, et les lecteurs d'écran peuvent ne pas lire le contenu masqué.
- **Autres possibilités** : Pour des situations où vous souhaitez masquer un élément mais le garder dans le flux du document, envisagez d'utiliser **visibility: hidden;** ou des techniques de positionnement.

En résumé, **display: none;** est un outil puissant pour contrôler la visibilité et la présence des éléments dans la mise en page d'une page web. Il doit être utilisé judicieusement pour s'assurer qu'il répond aux besoins de la conception sans nuire à l'expérience utilisateur globale.

IV - LA MISE EN PAGE

1) LE POSITIONNEMENT

Le positionnement en CSS est un aspect crucial pour contrôler la disposition des éléments sur une page web. Il existe plusieurs valeurs pour la propriété **position** qui déterminent comment un élément est positionné. Voici un aperçu concis des différents types de positionnement :

A - STATIC

- **Comportement** : C'est le positionnement par défaut. L'élément est positionné selon le flux normal du document.
- **Particularité** : Les propriétés **top, right, bottom, left** et **z-index** n'ont aucun effet sur un élément positionné en **static**.

B - RELATIVE

- **Comportement** : L'élément est positionné par rapport à sa position initiale dans le flux normal du document.
- **Particularité** : Vous pouvez utiliser **top, right, bottom, left** pour le déplacer par rapport à sa position originale. L'élément reste dans le flux du document.

C - ABSOLUTE

- **Comportement** : L'élément est retiré du flux normal du document et positionné par rapport à son ancêtre positionné le plus proche (non **static**) ou, à défaut, par rapport au bloc conteneur initial.
- **Particularité** : Il peut être positionné précisément avec **top, right, bottom, left**. Il ne prend pas d'espace dans le flux normal du document.

D - FIXED

- **Comportement** : L'élément est retiré du flux normal du document et positionné par rapport à la fenêtre du navigateur.
- **Particularité** : Il reste en place même lors du défilement de la page. Utilisez **top, right, bottom, left** pour le positionner.

E - STICKY

- **Comportement** : Un mélange entre **relative** et **fixed**. L'élément est traité comme **relative** jusqu'à ce qu'il atteigne un certain point lors du défilement, puis il devient **fixed**.
- **Particularité** : Nécessite l'utilisation de **top, right, bottom, left** pour définir le point de basculement.

F - UNSET

- **Comportement** : Annule le positionnement spécifique et revient au comportement par défaut du navigateur.
- **Particularité** : Utile pour réinitialiser le style d'un élément hérité.

G - POINTS ESSENTIELS

- **Choix du Positionnement** : Le choix du type de positionnement dépend de l'effet désiré dans la mise en page.
- **Impact sur le Flux du Document** : **absolute** et **fixed** retirent l'élément du flux normal, tandis que **static**, **relative**, et **sticky** le maintiennent dans le flux.
- **Utilisation avec d'Autres Propriétés** : Les propriétés **top, right, bottom, left**, et **z-index** sont souvent utilisées en conjonction avec le positionnement pour contrôler précisément l'emplacement des éléments.

En comprenant et en utilisant correctement ces différents types de positionnement, vous pouvez créer des mises en page complexes et réactives pour vos pages web.

2) LA POSITION ABSOLUTE

La propriété **position: absolute;** en CSS est utilisée pour positionner un élément de manière absolue par rapport à son conteneur le plus proche ayant une position non statique (c'est-à-dire **relative**, **absolute**, **fixed**, ou **sticky**). Lorsqu'un élément est positionné de manière absolue, il est retiré du flux normal du document, ce qui signifie qu'il n'affecte pas la position des autres éléments sur la page.

A - FONCTIONNEMENT DE POSITION: ABSOLUTE;

- **Positionnement :** L'élément est positionné par rapport aux bords de son conteneur le plus proche positionné. Si aucun conteneur positionné n'est trouvé, il se positionne par rapport au corps du document (**<body>**).
- **Décalages :** Les propriétés **top**, **right**, **bottom**, et **left** sont utilisées pour déplacer l'élément par rapport à son conteneur. Par exemple, **top: 20px;** déplacera l'élément de 20 pixels vers le bas à partir du bord supérieur de son conteneur.
- **Superposition :** La propriété **z-index** peut être utilisée pour contrôler la superposition de l'élément par rapport à d'autres éléments positionnés.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Position absolute</title>
6      <style>
7          .container {
8              position: relative; /* Conteneur positionné pour l'élément absolument positionné */
9          }
10
11         .absolue {
12             position: absolute;
13             top: 10px;
14             left: 10px;
15         }
16     </style>
17 </head>
18 <body>
19     <div class="container">
20         <h1>Position absolute</h1>
21         <div class="absolue">Je suis absolument positionné!</div>
22         <!-- Le reste de la page -->
23     </div>
24 </body>
25 </html>

```

Dans cet exemple, l'élément **.absolue** est positionné absolument à 10 pixels du bord supérieur et gauche de son conteneur **.container**.

B - POINTS ESSENTIELS

- **Conteneur Positionné** : Pour que **position: absolute;** fonctionne comme prévu, l'élément doit avoir un ancêtre positionné (autre que **static**). Si aucun n'est trouvé, l'élément sera positionné par rapport au corps du document.
- **Retrait du Flux Normal** : Un élément positionné absolument ne prend pas de place dans le flux normal du document. Cela signifie que les autres éléments se comporteront comme s'il n'existe pas.
- **Utilisation Prudente** : Bien que puissant, le positionnement absolu doit être utilisé avec prudence, car il peut entraîner des mises en page complexes et difficiles à gérer, en particulier sur des écrans de tailles différentes.

C - EN BREF

Position: absolute; est un outil puissant pour le positionnement précis des éléments, mais il nécessite une compréhension claire de son fonctionnement et de son impact sur la mise en page globale.

3) POSITION ABSOLUTE ET VALEUR NEGATIVE

L'utilisation conjointe de la position **absolute** avec des valeurs négatives sur les propriétés **top**, **left**, **right**, et **bottom** permet de positionner un élément en dehors des limites habituelles de son conteneur. Cette technique est souvent utilisée pour créer des effets de débordement ou pour positionner précisément un élément par rapport à un autre.

A - FONCTIONNEMENT

Lorsqu'un élément est positionné de manière absolue (**position: absolute;**), il est placé par rapport à son ancêtre positionné le plus proche (un élément avec une propriété **position** définie sur **relative**, **absolute**, **fixed**, ou **sticky**). En utilisant des valeurs négatives pour **top**, **left**, **right**, ou **bottom**, l'élément peut être déplacé au-delà des limites de ce conteneur.

B - DEUX EXEMPLES D'UTILISATION

1. Débordement vers le haut ou la gauche

Si vous définissez **top: -10px**; et/ou **left: -10px**; , l'élément se déplacera de 10 pixels vers le haut ou vers la gauche, respectivement, en dehors de son conteneur.

```
.element {
    position: absolute;
    top: -10px;
    left: -10px;
}
```

2. Débordement vers le bas ou la droite

De même, en utilisant **bottom: -10px**; et/ou **right: -10px**; , l'élément se déplacera de 10 pixels vers le bas ou vers la droite, au-delà de son conteneur.

```
.element {
    position: absolute;
    bottom: -10px;
    right: -10px;
}
```

C - POINTS ESSENTIELS

- **Impact sur la mise en page** : Comme l'élément est positionné en dehors de son conteneur, cela peut affecter la manière dont le reste du contenu est affiché ou interagit avec cet élément.
- **Attention aux débordements** : Il faut être prudent avec les débordements, surtout sur les petits écrans ou dans des conteneurs à taille variable, car cela peut entraîner des problèmes de défilement ou de contenu coupé.
- **Utilisation avec z-index** : L'utilisation de **z-index** peut être nécessaire pour contrôler la superposition de l'élément par rapport aux autres éléments de la page.

D - EN BREF

L'utilisation de valeurs négatives avec **position: absolute;** est une technique avancée qui permet de créer des effets de mise en page spécifiques, mais elle doit être utilisée avec prudence pour s'assurer qu'elle ne perturbe pas l'expérience utilisateur globale.

4) POSITION ABSOLUTE ET Z-INDEX

L'utilisation conjointe de **position: absolute;** et de **z-index** en CSS est une technique puissante pour contrôler le positionnement précis et la superposition d'éléments sur une page web. Voici un aperçu détaillé de leur interaction et de leur utilisation :

A - Z-INDEX

- **Superposition** : **z-index** détermine l'ordre de superposition des éléments. Des valeurs plus élevées placent l'élément plus "haut" dans la pile, le rendant visuellement au-dessus des éléments avec un **z-index** inférieur.
- **Contexte de superposition** : **z-index** ne fonctionne que sur des éléments positionnés (pas **static**). Le contexte de superposition est créé par l'élément positionné le plus proche, ce qui signifie que **z-index** fonctionne relativement à ce conteneur.

B - UTILISATION CONJOINTE

Lorsque vous utilisez **position: absolute;** avec **z-index**, vous pouvez non seulement positionner un élément précisément mais aussi contrôler sa superposition par rapport aux autres éléments positionnés.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Z-index</title>
6      <style>
7          .container {
8              position: relative;
9          }
10
11         .box {
12             position: absolute;
13             width: 100px;
14             height: 100px;
15         }
16
17         .box1 {
18             left: 10px;
19             top: 10px;
20             z-index: 2; /* Box 1 sera au-dessus de Box 2 */
21         }
22
23         .box2 {
24             left: 30px;
25             top: 30px;
26             z-index: 1; /* Box 2 sera en dessous de Box 1 */
27         }
28     </style>
29 </head>
30 <body>
31     <div class="container">
32         <h1>Z-index</h1>
33         <div class="box box1">Box 1</div>
34         <div class="box box2">Box 2</div>
35         <!-- Le reste de la page -->
36     </div>
37 </body>
38 </html>

```

Dans cet exemple, **box1** et **box2** sont tous deux positionnés absolument par rapport à leur conteneur **.container**. Grâce à **z-index**, **box1** apparaîtra au-dessus de **box2**, même si **box2** est déclaré après dans le HTML.

D - POINTS ESSENTIELS

- **Contexte de superposition** : Il est crucial de comprendre que **z-index** fonctionne dans le contexte de chaque pile de superposition. Deux éléments avec des ancêtres positionnés différents peuvent avoir un comportement de superposition différent.
- **Compatibilité navigateur** : Bien que largement supporté, le comportement exact de **z-index** peut varier légèrement entre les navigateurs, surtout dans des situations complexes.
- **Performance** : L'utilisation excessive de **position: absolute;** et de **z-index** peut rendre la mise en page plus difficile à gérer et peut affecter les performances, en particulier sur des appareils plus anciens ou moins puissants.

E - EN BREF

L'association de **position: absolute;** et de **z-index** est un outil puissant pour les développeurs web, permettant un contrôle précis du positionnement et de la superposition des éléments. Cependant, il nécessite une compréhension claire de la façon dont les éléments interagissent dans le contexte de la pile de superposition.

5) POSITION FIXED

La propriété **position: fixed;** en CSS est utilisée pour positionner un élément de manière fixe par rapport à la fenêtre du navigateur. Cela signifie que l'élément reste à la même place dans la fenêtre du navigateur, même lorsque l'utilisateur fait défiler la page. C'est une technique couramment utilisée pour des éléments tels que les barres de navigation, les boutons de retour en haut de page, ou les panneaux d'information qui doivent rester visibles à tout moment.

A - FONCTIONNEMENT DE POSITION: FIXED;

- **Positionnement** : L'élément est retiré du flux normal du document et positionné par rapport à la fenêtre du navigateur.
- **Décalages** : Les propriétés **top**, **right**, **bottom**, et **left** sont utilisées pour positionner l'élément par rapport aux bords de la fenêtre du navigateur.
- **Superposition** : La propriété **z-index** peut être utilisée pour contrôler la superposition de l'élément par rapport aux autres éléments de la page.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Fixed</title>
6      <style>
7          .fixed-element {
8              position: fixed;
9              bottom: 10px;
10             right: 10px;
11             width: 200px;
12             background-color: white;
13             border: 1px solid black;
14             padding: 10px;
15             z-index: 100;
16         }
17     </style>
18 </head>
19 <body>
20     <h1>Fixed</h1>
21     <div class="fixed-element">Je suis fixé!</div>
22     <!-- Le reste de la page -->
23 </body>
24 </html>

```

Dans cet exemple, **.fixed-element** est positionné en bas à droite de la fenêtre du navigateur. Il restera à cet endroit même lorsque l'utilisateur fait défiler la page.

B - POINTS ESSENTIELS

- Visibilité constante : `position: fixed;`** est idéal pour les éléments qui doivent rester visibles à tout moment, comme les barres de navigation ou les boutons d'action.
- Impact sur le flux du document :** Comme avec `position: absolute;`, un élément avec `position: fixed;` est retiré du flux normal du document, ce qui signifie qu'il n'affecte pas la position des autres éléments.
- Compatibilité mobile :** Sur les appareils mobiles, le comportement de `position: fixed;` peut varier, notamment en raison des différences dans la gestion du défilement. Il est important de tester son comportement sur différents appareils.
- Attention aux recouvrements :** Les éléments fixés peuvent recouvrir d'autres parties du contenu, il est donc important de les utiliser judicieusement pour ne pas gêner l'expérience utilisateur.

C - EN BREF

Position: `fixed;` est un outil puissant pour maintenir la visibilité d'éléments spécifiques lors du défilement de la page. Son utilisation doit cependant être équilibrée avec les besoins globaux de la mise en page et de l'expérience utilisateur.

6) POSITION STICKY

La propriété **position: sticky;** en CSS est une combinaison des comportements des propriétés **relative** et **fixed**. Elle permet à un élément de se comporter comme un élément **position: relative** dans le contexte normal du flux de la page, puis de devenir **fixed** lorsqu'il atteint un certain point lors du défilement.

A - FONCTIONNEMENT DE POSITION: STICKY;

- **Positionnement initial** : Initialement, l'élément se comporte comme s'il était en **position: relative**. Il suit le flux normal du document.
- **Activation du sticky** : Lorsque l'utilisateur fait défiler la page et atteint un certain point spécifié par **top, right, bottom**, ou **left**, l'élément "colle" à cette position dans la fenêtre du navigateur.
- **Décalages** : Les propriétés **top, right, bottom**, et **left** déterminent le point de déclenchement où l'élément devient fixe. Par exemple, **top: 10px;** signifie que l'élément deviendra fixe une fois que 10px du haut de l'élément auront été défilés hors de la vue.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Sticky</title>
6      <style>
7          .sticky-element {
8              position: -webkit-sticky; /* Pour la compatibilité avec Safari */
9              position: sticky;
10             top: 20px; /* L'élément devient sticky à 20px du haut de la fenêtre */
11             background-color: yellow;
12             padding: 10px;
13             border: 1px solid black;
14         }
15     </style>
16 </head>
17 <body>
18     <h1>Sticky</h1>
19     <!-- Une partie de la page de la page -->
20     <div class="sticky-element">Je suis sticky!</div>
21     <!-- Le reste de la page -->
22 </body>
23 </html>

```

Dans cet exemple, **.sticky-element** commence par se comporter normalement. Lorsque l'utilisateur fait défiler la page et que le haut de **.sticky-element** atteint 20px du haut de la fenêtre du navigateur, l'élément devient fixe à cette position.

B - POINTS ESSENTIELS

- **Compatibilité navigateur** : **position: sticky;** est relativement bien supporté dans les navigateurs modernes, mais il peut y avoir des problèmes de compatibilité avec certains navigateurs plus anciens, notamment Safari (d'où l'utilisation de **-webkit-sticky**).
- **Contexte de superposition** : Comme avec **fixed** et **absolute**, **sticky** peut utiliser **z-index** pour gérer la superposition avec d'autres éléments.

- **Limitations :** `sticky` ne fonctionnera pas correctement si l'ancêtre le plus proche avec une propriété `overflow` autre que `visible` est rencontré.
- **Utilisation pratique :** Cette propriété est particulièrement utile pour des éléments comme les en-têtes de tableaux ou les barres de navigation qui doivent rester visibles pendant le défilement.

C - EN BREF

Position: `sticky`; offre une solution flexible pour des éléments qui doivent être positionnés de manière relative dans le flux normal du document, mais qui doivent également rester visibles à un certain point lors du défilement. Cela en fait un outil précieux pour améliorer l'expérience utilisateur sans perturber la mise en page générale.

7) INLINE-BLOCK, UN COMPORTEMENT HYBRIDE

La propriété CSS `display: inline-block;` combine certains aspects des modes de boîte `inline` et `block`. Elle permet de structurer la mise en page avec plus de flexibilité que ces deux modes pris séparément.

A - PRINCIPE DE FONCTIONNEMENT

- **Comportement inline :** Comme avec `display: inline;`, les éléments avec `display: inline-block;` se placent sur la même ligne les uns à côté des autres, plutôt que sur de nouvelles lignes, comme c'est le cas avec `display: block;`.
- **Dimensions réglables :** Contrairement aux éléments `inline`, les éléments `inline-block` peuvent avoir des largeurs (`width`) et hauteurs (`height`) explicites.
- **Marges et espacements :** Les éléments `inline-block` acceptent les marges (`margin`) et les espacements internes (`padding`) sur tous les côtés, sans casser le flux de la ligne, ce qui est une limitation des éléments purement `inline`.

B - UTILISATION

1. **Mise en page de composants côté à côté :** `inline-block` est idéal pour placer des éléments côté à côté, comme des boutons ou des cartes, tout en conservant la capacité de définir leur taille et leur espacement.
2. **Création de grilles légères :** Avant l'avènement de Flexbox et Grid, `inline-block` était souvent utilisé pour créer des grilles simples, car il permet de placer des éléments en lignes et colonnes avec un contrôle sur les dimensions.
3. **Menus de navigation :** Les éléments de navigation, comme les listes dans une barre de navigation, peuvent être mis en œuvre avec `inline-block` pour les aligner horizontalement tout en contrôlant l'espacement et l'alignement.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Inline-block</title>
6      <style>
7          .inline-block-element {
8              display: inline-block;
9              width: 100px;
10             height: 100px;
11             margin: 10px;
12             padding: 5px;
13             border: 1px solid black;
14         }
15     </style>
16 </head>
17 <body>
18     <h1>Inline-block</h1>
19     <div class="container">
20         <div class="inline-block-element">Élément 1</div>
21         <div class="inline-block-element">Élément 2</div>
22         <div class="inline-block-element">Élément 3</div>
23     </div>
24     <!-- Le reste de la page -->
25 </body>
26 </html>

```

Dans cet exemple, chaque **.inline-block-element** sera affiché côte à côte, avec des dimensions spécifiques, des marges et des espacements internes.

C - POINTS ESSENTIELS

- Flux du document :** **inline-block** ne perturbe pas le flux de ligne, permettant une intégration plus fluide avec le texte et d'autres éléments inline.
- Espacement vertical :** Les éléments **inline-block** peuvent parfois présenter des espaces inattendus, en particulier verticalement, dus à l'alignement de la ligne de base. Ce comportement peut nécessiter des ajustements.
- Alternative à flexbox et grid :** Avec l'arrivée de Flexbox et Grid, l'utilisation de **inline-block** pour des mises en page complexes a diminué, mais il reste utile pour des cas d'utilisation simples et spécifiques.

D - EN BREF

Display: inline-block; est un outil utile pour les mises en page qui nécessitent des éléments alignés horizontalement avec un contrôle complet sur leur taille et leur espacement, tout en restant dans le flux normal du document.

8) LES TABLES AVEC DISPLAY

La propriété CSS **display: table;** est utilisée pour imiter le comportement d'une table HTML (**<table>**) en utilisant des éléments CSS. Elle est souvent utilisée en combinaison avec **display: table-row;** et **display: table-cell;** pour créer des mises en page qui ressemblent à des tableaux, mais sans utiliser les balises de tableau HTML traditionnelles.

A - PRINCIPE DE FONCTIONNEMENT

- **Simulation de Tableau HTML :** **display: table;** permet à un élément de se comporter comme l'élément **<table>** en HTML. Cela signifie que l'élément agira comme un conteneur pour des éléments de ligne et de cellule.
- **Structure de Tableau :** Pour créer une structure complète de tableau, vous pouvez utiliser **display: table-row;** pour simuler des lignes (**<tr>**) et **display: table-cell;** pour simuler des cellules (**<td>** ou **<th>**).

B - USAGE

1. **Mises en page de type tableau :** Lorsque vous avez besoin d'une mise en page qui ressemble à un tableau, mais que vous voulez éviter d'utiliser des balises de tableau HTML pour des raisons de sémantique ou de flexibilité, **display: table;** et ses variantes sont utiles.
2. **Alignement vertical :** Un avantage clé de l'utilisation de **display: table-cell;** est la possibilité d'aligner verticalement le contenu à l'intérieur d'un élément, ce qui est difficile à réaliser avec d'autres valeurs de **display**.
3. **Compatibilité navigateur :** Cette méthode offre une bonne compatibilité avec les anciens navigateurs qui ne prennent pas en charge Flexbox ou Grid.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Display – Table</title>
6      <style>
7          .table {
8              display: table;
9              border-collapse: collapse;
10         }
11
12         .table-row {
13             display: table-row;
14         }
15
16         .table-cell {
17             display: table-cell;
18             border: 1px solid black;
19             padding: 10px;
20         }
21     </style>
22 </head>
23 <body>
24     <h1>Display – Table1</h1>
25     <div class="table">
26         <div class="table-row">
27             <div class="table-cell">Cellule 1</div>
28             <div class="table-cell">Cellule 2</div>
29         </div>
30         <div class="table-row">
31             <div class="table-cell">Cellule 3</div>
32             <div class="table-cell">Cellule 4</div>
33         </div>
34     </div>
35     <!-- Le reste de la page -->
36 </body>
37 </html>
```

Dans cet exemple, **.table** agit comme un tableau, chaque **.table-row** comme une ligne de tableau, et chaque **.table-cell** comme une cellule de tableau.

C - POINTS ESSENTIELS

- Non destiné aux données tabulaires** : Bien que **display: table**; imite le comportement d'un tableau, il ne doit pas être utilisé pour afficher des données tabulaires réelles. Pour cela, l'élément **<table>** est plus approprié en raison de sa sémantique.
- Flexibilité limitée** : Bien que cette approche offre un certain degré de flexibilité, elle est moins puissante que Flexbox ou Grid pour des mises en page complexes.

- **Utilisation pour l'alignement :** L'un des usages les plus courants de **display: table-cell;** est l'alignement vertical, qui peut être difficile à réaliser avec d'autres méthodes.

D - EN BREF

Display: table; et ses variantes sont utiles pour créer des mises en page qui nécessitent un alignement similaire à celui des tableaux, tout en offrant plus de flexibilité que les tableaux HTML traditionnels. Cependant, avec l'avènement de Flexbox et Grid, l'utilisation de **display: table;** pour des mises en page complexes a diminué.

9) FLEXBOX - VUE D'ENSEMBLE

Flexbox, ou le modèle de boîte flexible, est un outil puissant en CSS qui permet de créer des mises en page efficaces et flexibles. Il est particulièrement utile pour aligner des éléments de manière prévisible, répartir l'espace entre les éléments d'un conteneur, et gérer la taille dynamique des éléments.

A - PRINCIPES DE BASE DE FLEXBOX

1. **Conteneur flex :** Pour utiliser Flexbox, vous devez d'abord définir un conteneur flex en utilisant **display: flex;** ou **display: inline-flex;**. Ce conteneur devient le contexte flex pour tous ses enfants directs.

```
.container {
  display: flex;
}
```

2. **Éléments flex :** Les enfants directs d'un conteneur flex sont appelés éléments flex. Ils peuvent être manipulés de diverses manières à l'intérieur du conteneur.

3. **Direction Principale :** La propriété **flex-direction** définit la direction principale de l'axe dans le conteneur flex (par exemple, **row** ou **column**).

```
.container {
  flex-direction: row; /* ou column */
}
```

4. **Alignement et justification :** Utilisez **align-items** pour aligner les éléments flex verticalement et **justify-content** pour les aligner horizontalement.

```
.container {
  align-items: center; /* alignement vertical */
  justify-content: space-between; /* alignement horizontal */
}
```

5. **Flexibilité des éléments :** La propriété **flex** sur les éléments flexibles contrôle leur capacité à grandir ou à rétrécir par rapport aux autres éléments flexibles dans le conteneur. Elle est souvent définie par **flex: [flex-grow] [flex-shrink] [flex-basis]**.

```
.item {
  flex: 1 1 auto; /* grandit et rétrécit avec un point de départ automatique */
}
```

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Flexbox</title>
6      <style>
7          .container {
8              display: flex;
9              justify-content: space-around;
10             align-items: center;
11         }
12
13         .item {
14             flex: 1;
15         }
16     </style>
17 </head>
18 <body>
19     <h1>Flexbox</h1>
20     <div class="container">
21         <div class="item">Item 1</div>
22         <div class="item">Item 2</div>
23         <div class="item">Item 3</div>
24     </div>
25     <!-- Le reste de la page -->
26 </body>
27 </html>

```

Dans cet exemple, les **.item** sont répartis uniformément dans le **.container**, centrés verticalement, et chacun prend une part égale de l'espace disponible.

B - AVANTAGES DE FLEXBOX

- Flexibilité** : Flexbox permet une gestion plus flexible des mises en page, en particulier pour les interfaces réactives et les applications web.
- Alignement facile** : L'alignement des éléments, qui peut être complexe avec les méthodes traditionnelles, est simplifié avec Flexbox.
- Gestion de l'espace** : La répartition de l'espace entre les éléments est plus intuitive et moins sujette aux erreurs.

C - LIMITATIONS

- **Compatibilité navigateur** : Bien que largement supporté maintenant, Flexbox peut présenter des problèmes dans les anciens navigateurs.
- **Apprentissage** : Comprendre toutes les subtilités de Flexbox peut nécessiter un certain temps d'apprentissage.

D - EN BREF

Flexbox est un outil extrêmement utile pour créer des mises en page modernes et réactives en CSS. Il offre une grande flexibilité et contrôle, rendant l'alignement et la distribution de l'espace entre les éléments beaucoup plus simples et plus intuitifs.

10) FLEXBOX - GESTION DES AXES

Flexbox offre une grande flexibilité dans la gestion des directions principales et secondaires des éléments dans un conteneur. Les propriétés clés pour contrôler ces directions sont **flex-direction**, **justify-content**, et **align-items**. Comprendre comment elles interagissent est essentiel pour maîtriser Flexbox.

A - FLEX DIRECTION

La propriété **flex-direction** définit l'axe principal du conteneur flex, déterminant la direction dans laquelle les éléments flexibles sont placés dans le conteneur.

- **Row** : **flex-direction: row;** aligne les éléments horizontalement, de gauche à droite.
- **Column** : **flex-direction: column;** aligne les éléments verticalement, de haut en bas.

B - JUSTIFY CONTENT

justify-content contrôle l'alignement des éléments le long de l'axe principal (défini par **flex-direction**).

- **Row** : Lorsque **flex-direction** est **row**, **justify-content** gère l'alignement horizontal.
 - **justify-content: flex-start;** aligne les éléments au début (gauche).
 - **justify-content: flex-end;** les aligne à la fin (droite).
 - **justify-content: center;** les centre horizontalement.
 - **justify-content: space-between;** répartit les éléments uniformément, avec le premier élément au début et le dernier à la fin.
 - **justify-content: space-around;** répartit les éléments uniformément avec un espace égal autour de chaque élément.
- **Column** : Lorsque **flex-direction** est **column**, **justify-content** gère l'alignement vertical.
 - Les mêmes valeurs (**flex-start**, **flex-end**, **center**, **space-between**, **space-around**) s'appliquent, mais elles contrôlent l'espacement vertical.

C - ALIGN ITEMS

align-items contrôle l'alignement des éléments le long de l'axe secondaire (perpendiculaire à l'axe principal).

- **Row** : Avec **flex-direction: row**, **align-items** gère l'alignement vertical.
 - **align-items: flex-start;** aligne les éléments en haut.
 - **align-items: flex-end;** les aligne en bas.
 - **align-items: center;** les centre verticalement.

- **align-items: stretch;** (valeur par défaut) étire les éléments pour remplir le conteneur en hauteur.
- **Column :** Avec **flex-direction: column, align-items** gère l'alignement horizontal.
 - Les mêmes valeurs (**flex-start, flex-end, center, stretch**) s'appliquent, mais elles contrôlent l'alignement horizontal.

D - EXEMPLE UTILISANT CES PROPRIÉTÉS

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Flexbox</title>
6      <style>
7          .container {
8              display: flex;
9              flex-direction: row; /* ou column */
10             justify-content: center; /* alignement principal */
11             align-items: center; /* alignement secondaire */
12         }
13
14         .item {
15             width: 50px;
16             height: 50px;
17         }
18     </style>
19 </head>
20 <body>
21     <h1>Flexbox</h1>
22     <div class="container">
23         <div class="item">Item 1</div>
24         <div class="item">Item 2</div>
25         <div class="item">Item 3</div>
26     </div>
27     <!-- Le reste de la page -->
28 </body>
29 </html>

```

Dans cet exemple, en jouant avec **flex-direction, justify-content**, et **align-items**, vous pouvez contrôler précisément comment les éléments **.item** sont disposés et alignés dans le **.container**.

D - EN BREF

Flex-direction détermine l'orientation principale des éléments dans un conteneur flex, tandis que **justify-content** et **align-items** offrent un contrôle détaillé sur leur alignement le long des axes principal et secondaire, respectivement. Cette combinaison permet de créer des mises en page flexibles et réactives adaptées à de nombreux besoins de design.

11) FLEXBOX - AUTRES POINTS IMPORTANTS

flex-wrap, **flex-shrink**, et **flex-grow** sont des propriétés essentielles du modèle de boîte flexible (Flexbox) en CSS, offrant un contrôle avancé sur le comportement des éléments flexibles dans un conteneur flex. Comprendre ces propriétés est crucial pour créer des mises en page réactives et flexibles.

A - FLEX WRAP

La propriété **flex-wrap** contrôle si les éléments flexibles sont forcés de rester sur une seule ligne ou peuvent s'enrouler sur plusieurs lignes.

- **nowrap** (valeur par défaut) : Les éléments flexibles sont disposés sur une seule ligne, ce qui peut entraîner un débordement si l'espace n'est pas suffisant.

```
.container {
  flex-wrap: nowrap;
}
```

- **wrap** : Les éléments flexibles s'enroulent sur plusieurs lignes, du haut vers le bas, si l'espace n'est pas suffisant sur la ligne actuelle.

```
.container {
  flex-wrap: wrap;
}
```

- **wrap-reverse** : Semblable à **wrap**, mais les éléments s'enroulent en sens inverse, du bas vers le haut.

```
.container {
  flex-wrap: wrap-reverse;
}
```

B - FLEX SHRINK

La propriété **flex-shrink** définit la capacité d'un élément à rétrécir si nécessaire. Elle prend un nombre comme valeur, représentant le "facteur de rétrécissement".

- **0** : L'élément ne rétrécira pas, même s'il manque d'espace.
- **1** (valeur par défaut) : L'élément peut rétrécir si nécessaire pour ne pas déborder du conteneur.

```
.item {
  flex-shrink: 1; /* ou tout autre nombre */
}
```

C - FLEX GROW

La propriété **flex-grow** définit la capacité d'un élément à grandir pour remplir l'espace disponible dans le conteneur. Elle prend également un nombre comme valeur, représentant le "facteur de croissance".

- **0** (valeur par défaut) : L'élément ne grandira pas pour remplir l'espace disponible.
- **1** : L'élément grandira pour occuper l'espace disponible. Si plusieurs éléments ont **flex-grow** défini sur 1, l'espace disponible sera réparti également entre eux.

```
.item {
  flex-grow: 1; /* ou tout autre nombre */
}
```

D - EXEMPLE UTILISANT CES PROPRIÉTÉS

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8">
5    <title>Flexbox</title>
6    <style>
7      .container {
8        display: flex;
9        /* Les éléments peuvent s'enrouler sur plusieurs lignes */
10       flex-wrap: wrap;
11     }
12
13     .item {
14       width: 200px; /* Largeur de départ */
15     }
16
17     .item1 {
18       /* Cet élément grandira pour remplir l'espace disponible */
19       flex-grow: 1;
20       /* Cet élément peut rétrécir plus que les autres si nécessaire */
21       flex-shrink: 2;
22     }
23
24     .item2 {
25       /* Cet élément grandira deux fois plus que item1 */
26       flex-grow: 2;
27       /* Cet élément peut rétrécir normalement */
28       flex-shrink: 1;
29     }
30
31     .item3 {
32       /* Cet élément ne grandira pas pour remplir l'espace disponible */
33       flex-grow: 0;
34       /* Cet élément ne rétrécira pas non plus */
35       flex-shrink: 0;
36     }
37   </style>
38 </head>
39 <body>
40   <h1>Flexbox</h1>
41   <div class="container">
42     <div class="item">Item 1</div>
43     <div class="item">Item 2</div>
44     <div class="item">Item 3</div>
45   </div>
46   <!-- Le reste de la page -->
47 </body>
48 </html>
```

Dans cet exemple :

- **Item 1** a un **flex-grow** de 1 et un **flex-shrink** de 2, ce qui signifie qu'il grandira pour remplir l'espace disponible, mais rétrécira également plus que les autres si l'espace est restreint.
- **Item 2** a un **flex-grow** de 2, lui permettant de grandir deux fois plus que l'Item 1. Son **flex-shrink** est de 1, donc il rétrécira normalement.
- **Item 3** ne grandira ni ne rétrécira, car ses valeurs de **flex-grow** et **flex-shrink** sont toutes deux fixées à 0.

Ce scénario montre comment les éléments flexibles peuvent se comporter différemment dans un conteneur flex, en fonction de leurs propriétés **flex-grow** et **flex-shrink**. Cela permet une grande flexibilité dans la conception de mises en page réactives.

E - EN BREF

Flex-wrap, **flex-shrink**, et **flex-grow** offrent un contrôle détaillé sur la manière dont les éléments flexibles se comportent en termes de rétrécissement, de croissance et de disposition sur plusieurs lignes, permettant ainsi de créer des mises en page flexibles et adaptatives.

12) LES GRILLES EN CSS (GRID LAYOUT)

Le Grid Layout (ou système de grille) en CSS est un puissant outil de mise en page qui permet de créer des interfaces complexes et réactives de manière intuitive et flexible. Il s'agit d'un système bidimensionnel, contrairement à Flexbox qui est principalement unidimensionnel, ce qui signifie que vous pouvez contrôler la disposition des éléments à la fois horizontalement et verticalement.

A - PRINCIPES DE BASE DU GRID LAYOUT

1. **Définition d'un conteneur grid** : Pour utiliser le Grid Layout, vous devez d'abord définir un conteneur grid en utilisant **display: grid;** ou **display: inline-grid;**

```
.container {
  display: grid;
}
```

2. **Définition des colonnes et des lignes** : Utilisez **grid-template-columns** et **grid-template-rows** pour définir la taille des colonnes et des lignes dans la grille.

```
.container {
  grid-template-columns: 100px 200px auto;
  grid-template-rows: 50px auto 100px;
}
```

Dans cet exemple, le conteneur grid aura trois colonnes de largeurs 100px, 200px, et auto, et trois lignes de hauteurs 50px, auto, et 100px.

3. **Placement des éléments grid** : Placez les éléments enfants dans la grille en utilisant **grid-column** et **grid-row**.

```
.item {
  grid-column: 1 / 3;
  grid-row: 2 / 4;
}
```

Cela place **.item** pour qu'il s'étende de la première à la troisième colonne et de la deuxième à la quatrième ligne.

4. **Espacement entre les éléments** : **grid-gap** (ou **gap** dans les versions plus récentes de CSS) définit l'espace entre les éléments de la grille.

```
.container {
    gap: 10px;
}
```

5. **Alignement et justification** : Utilisez **align-items**, **justify-items**, **align-content**, et **justify-content** pour aligner et justifier les éléments et le contenu dans la grille.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Grid</title>
6      <style>
7          .container {
8              display: grid;
9              grid-template-columns: repeat(3, 1fr);
10             grid-template-rows: auto;
11             gap: 10px;
12         }
13
14         .item {
15             background-color: lightblue;
16             border: 1px solid black;
17             text-align: center;
18         }
19     </style>
20 </head>
21 <body>
22     <h1>Grid</h1>
23     <div class="container">
24         <div class="item">Item 1</div>
25         <div class="item">Item 2</div>
26         <div class="item">Item 3</div>
27     </div>
28     <!-- Le reste de la page -->
29 </body>
30 </html>
```

Dans cet exemple, le **.container** est un conteneur grid avec trois colonnes de taille égale. Les **.item** sont placés automatiquement dans la grille.

B - AVANTAGES DU GRID LAYOUT

- **Contrôle bidimensionnel** : Grid permet de contrôler à la fois les colonnes et les lignes, offrant une flexibilité incroyable pour des mises en page complexes.
- **Disposition précise** : Vous pouvez placer avec précision les éléments où vous le souhaitez dans la grille.
- **Réactivité** : Grid Layout facilite la création de designs réactifs, en particulier pour les mises en page complexes.

C - LIMITATION

- **Compatibilité navigateur** : Bien que largement supporté maintenant, certains anciens navigateurs ne prennent pas en charge toutes les fonctionnalités de Grid Layout.

D - EN BREF

Le Grid Layout est un outil puissant pour créer des mises en page structurées et complexes en CSS. Il offre un contrôle précis et une grande flexibilité, rendant la conception de mises en page réactives et sophistiquées plus accessible et plus intuitive.

13) LES MEDIA QUERIES - PRINCIPE GENERAL

Les media queries en CSS sont extrêmement flexibles, permettant non seulement de cibler des dispositifs en fonction de leur taille d'écran pour adapter les mises en page aux différents appareils, comme les ordinateurs de bureau, les tablettes et les smartphones ; mais aussi de leur type (comme les écrans ou les imprimantes) et d'autres caractéristiques. Vous pouvez également combiner plusieurs conditions pour un ciblage plus précis.

A - DIFFERENCE ENTRE SCREEN ET PRINT

Les media queries permettent de distinguer entre différents types de médias, les plus courants étant **screen** (pour les écrans d'ordinateurs, tablettes, smartphones, etc.) et **print** (pour l'impression).

```

1  @media screen and (min-width: 600px) {
2    body {
3      background-color: lightblue;
4    }
5  }
6
7  @media print {
8    body {
9      color: black;
10     background-color: white;
11    }
12 }
```

Dans cet exemple, le **background-color** du **body** sera appliqué uniquement lorsque la media query est visualisée sur un écran et que la largeur de la fenêtre est d'au moins 600 pixels. Lors de l'impression du document, le fond et la couleur du texte changent. Par exemple, cela peut être utile pour assurer que le texte est lisible et que les arrière-plans colorés ne consomment pas d'encre inutilement.

B - UTILISATION DE PLUSIEURS CONDITIONS

Les media queries peuvent combiner plusieurs conditions pour un ciblage plus spécifique. Vous pouvez utiliser des opérateurs logiques tels que **and**, **,** (qui fonctionne comme un **or**), **not**, et **only**.

Exemple avec plusieurs conditions :

```
1 @media screen and (min-width: 600px) and (max-width: 1200px) {
2   .container {
3     padding: 20px;
4   }
5 }
```

Dans cet exemple, le style pour **.container** s'appliquera uniquement sur les écrans dont la largeur est comprise entre 600 et 1200 pixels.

C - COMBINAISON DE PLUSIEURS MEDIA QUERIES

Vous pouvez également écrire plusieurs media queries séparées par des virgules pour appliquer des styles dans différents scénarios.

```
1 @media screen and (max-width: 600px), print {
2   .container {
3     background-color: lightyellow;
4   }
5 }
```

Ici, le **background-color** de **.container** sera **lightyellow** soit sur les écrans de moins de 600 pixels de large, soit lors de l'impression.

D - EN BREF

- **Type de Média** : Utilisez **screen**, **print**, et d'autres types pour cibler différents médias.
- **Conditions Multiples** : Combine plusieurs conditions dans une seule media query pour un ciblage précis.
- **Combinaison de Media Queries** : Utilisez des virgules pour combiner plusieurs media queries, agissant comme un opérateur **or**.

Ces fonctionnalités rendent les media queries extrêmement puissantes pour créer des designs réactifs et adaptatifs, assurant que votre site ou application web ait une apparence et une fonctionnalité optimales sur tous les dispositifs et dans toutes les situations.

14) LES MEDIA QUERIES - LE MOBILE FIRST

“Mobile First” est une approche de conception où vous commencez par créer des designs pour les écrans mobiles (plus petits), puis vous utilisez les media queries pour ajuster ces designs pour des écrans plus grands.

1. **Styles de base pour mobile** : Commencez par écrire des styles CSS qui ciblent les appareils mobiles. Ces styles sont votre point de départ et devraient être adaptés aux écrans plus petits.

2. **Utilisez des media queries pour des écrans plus grands** : Ensuite, utilisez les media queries pour ajouter ou modifier des styles pour des écrans plus grands.

```

1 .container {
2   width: 100%;
3   padding: 10px;
4 }
5
6 @media (min-width: 600px) {
7   .container {
8     width: 80%;
9     padding: 20px;
10  }
11 }
12
13 @media (min-width: 1000px) {
14   .container {
15     width: 60%;
16     padding: 30px;
17  }
18 }
```

Dans cet exemple, le **.container** a des styles de base adaptés aux mobiles. Lorsque la largeur de l'écran atteint 600 pixels ou plus, les styles sont ajustés pour un affichage optimal sur des tablettes ou des écrans de taille moyenne. À 1000 pixels ou plus, les styles sont à nouveau ajustés, cette fois pour les écrans d'ordinateur de bureau.

Les avantages du Mobile First :

- **Performance** : Les appareils mobiles ont souvent moins de ressources que les ordinateurs de bureau, donc en commençant par le mobile, vous vous assurez que votre site est optimisé pour la performance sur ces appareils.
- **Priorisation du contenu** : Cette approche vous oblige à vous concentrer sur l'essentiel, car l'espace sur les appareils mobiles est limité.
- **Meilleure expérience utilisateur sur mobile** : Comme de plus en plus de personnes accèdent à Internet via des appareils mobiles, une expérience utilisateur optimisée sur mobile est cruciale.

En résumé, l'approche “Mobile First” commence par optimiser pour les écrans les plus petits, garantissant ainsi une performance et une expérience utilisateur optimales sur les appareils mobiles, tout en permettant des ajustements faciles pour les écrans plus grands.

15) FLEXBOX ET MEDIA QUERIES

L'utilisation combinée de Flexbox et des media queries en CSS est une autre méthode efficace pour créer des mises en page réactives. Flexbox est idéal pour les mises en page linéaires (unidimensionnelles), où vous voulez contrôler la disposition des éléments soit en ligne (horizontalement) soit en colonne (verticalement). En combinant Flexbox avec des media queries, vous pouvez ajuster la disposition des éléments en fonction de la taille de l'écran ou d'autres caractéristiques du dispositif d'affichage.

A - EXEMPLE D'UTILISATION COMBINEE

Supposons que vous ayez une mise en page avec plusieurs éléments disposés horizontalement sur un grand écran, et que vous souhaitiez les empiler verticalement sur des écrans plus petits, comme ceux des téléphones mobiles.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Flexbox et Media Queries</title>
6      <style>
7          .container {
8              display: flex;
9              /* Disposition verticale sur les petits écrans */
10             flex-direction: column;
11             gap: 10px;
12         }
13
14         .item {
15             background-color: lightblue;
16             border: 1px solid black;
17             text-align: center;
18             /* Les éléments se répartissent l'espace disponible */
19             flex-grow: 1;
20         }
21
22         @media (min-width: 600px) {
23             .container {
24                 /* Disposition horizontale pour les grands écrans */
25                 flex-direction: row;
26             }
27         }
28     </style>
29 </head>
30 <body>
31     <h1>Flexbox et Media Queries</h1>
32     <div class="container">
33         <div class="item">Élément 1</div>
34         <div class="item">Élément 2</div>
35         <div class="item">Élément 3</div>
36     </div>
37     <!-- Le reste de la page -->
38 </body>
39 </html>
```

Dans cet exemple, le **.container** utilise Flexbox pour disposer les **.item** horizontalement. Cependant, lorsque la largeur de l'écran est inférieure à 600 pixels, la media query ajuste le **.container** pour que les éléments soient disposés verticalement, créant ainsi une meilleure expérience utilisateur sur les petits écrans.

B - AVANTAGES DE CETTE APPROCHE

1. **Adaptabilité** : Les éléments s'adaptent automatiquement à différentes tailles d'écran, offrant une expérience utilisateur optimale sur divers appareils.
2. **Simplicité et efficacité** : Flexbox simplifie la disposition des éléments, en particulier pour les mises en page linéaires, et les media queries permettent des ajustements faciles pour différentes tailles d'écran.
3. **Maintenabilité et flexibilité** : Cette approche rend le code plus facile à maintenir et à ajuster, car les modifications de mise en page pour différentes tailles d'écran sont centralisées dans les media queries.

C - CONSEILS POUR UNE MEILLEURE PRATIQUE

- **Pensez mobile first** : Commencez par concevoir pour les petits écrans, puis utilisez des media queries pour ajuster la mise en page pour les écrans plus grands.
- **Testez sur divers appareils** : Assurez-vous que votre mise en page fonctionne bien sur une gamme d'appareils pour garantir une expérience utilisateur cohérente.
- **Utilisez des unités flexibles** : L'utilisation d'unités flexibles comme **flex-grow** et **flex-shrink** permet aux éléments de s'adapter dynamiquement à l'espace disponible.

D - EN RESUME

En résumé, la combinaison de Flexbox et de media queries offre une méthode robuste et flexible pour créer des mises en page réactives qui s'adaptent bien à une variété de tailles d'écran, améliorant ainsi l'expérience utilisateur sur différents appareils.

16) GRID LAYOUT ET MEDIA QUERIES

L'utilisation combinée du Grid Layout et des media queries en CSS permet de créer des mises en page réactives et adaptatives, qui changent en fonction de la taille de l'écran ou d'autres caractéristiques du dispositif d'affichage. Cette combinaison est particulièrement puissante pour ajuster les mises en page complexes afin qu'elles fonctionnent bien sur des appareils de toutes tailles, des téléphones mobiles aux grands écrans d'ordinateur.

A - EXEMPLE D'UTILISATION COMBINEE

Imaginons que vous ayez un site web avec une mise en page à trois colonnes sur un grand écran, que vous souhaitez transformer en une seule colonne sur un écran de téléphone mobile.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Grid et Media Queries</title>
6      <style>
7          .container {
8              display: grid;
9              /* Une seule colonne sur les petits écrans */
10             grid-template-columns: 1fr;
11             gap: 10px;
12         }
13
14         .item {
15             background-color: lightblue;
16             border: 1px solid black;
17             text-align: center;
18         }
19
20         @media (min-width: 600px) {
21             .container {
22                 /* Trois colonnes de taille égale pour les plus grands écrans */
23                 grid-template-columns: 1fr 1fr 1fr;
24             }
25         }
26     </style>
27 </head>
28 <body>
29     <h1>Grid et Media Queries</h1>
30     <div class="container">
31         <div class="item">Colonne 1</div>
32         <div class="item">Colonne 2</div>
33         <div class="item">Colonne 3</div>
34     </div>
35     <!-- Le reste de la page -->
36 </body>
37 </html>

```

Dans cet exemple, le **.container** utilise par défaut une grille à trois colonnes. Cependant, lorsque la largeur de l'écran est inférieure à 600 pixels (ce qui est courant pour les écrans de téléphone mobile), la media query ajuste le **.container** pour qu'il ait une seule colonne. Cela signifie que les **.item** seront empilés verticalement au lieu d'être disposés horizontalement.

B - AVANTAGES DE CETTE APPROCHE

- Flexibilité** : Vous pouvez créer des mises en page qui s'adaptent dynamiquement à la taille de l'écran, améliorant l'expérience utilisateur sur une variété d'appareils.
- Contrôle Précis** : Le Grid Layout offre un contrôle précis sur la disposition des éléments, tandis que les media queries permettent d'ajuster cette disposition en fonction des conditions d'affichage.
- Maintenabilité** : Il est plus facile de maintenir et de modifier une mise en page en utilisant cette approche, car les ajustements pour différentes tailles d'écran sont centralisés dans les media queries.

C - CONSEILS POUR UNE MEILLEURE PRATIQUE

- **Commencez par Mobile** : Envisagez de commencer par une mise en page simple pour les petits écrans (approche Mobile First) et utilisez des media queries pour ajouter des complexités pour les écrans plus grands.
- **Testez sur Plusieurs Appareils** : Assurez-vous de tester votre mise en page sur une variété d'appareils pour garantir une expérience utilisateur cohérente.
- **Utilisez des Unités Relatives** : Pour une meilleure réactivité, utilisez des unités relatives comme `fr` et `%` pour les tailles de grille.

En combinant le Grid Layout avec des media queries, vous pouvez créer des designs qui non seulement semblent bons sur tous les appareils, mais qui sont également flexibles et faciles à maintenir.

V - PERFECTIONNEMENT

1) LES BALISES <AUDIO> ET <VIDEO>

Les balises **<audio>** et **<video>** en HTML5 permettent d'intégrer facilement des fichiers audio et vidéo dans des pages web. Ces balises offrent une alternative native aux plugins tiers comme Flash utilisées précédemment, rendant le contenu multimédia plus accessible et plus facile à manipuler avec des standards web.

A - BALISE <AUDIO>

La balise **<audio>** est utilisée pour intégrer un contenu audio dans une page web.

- Syntaxe de Base

```

1 <audio controls>
2   <source src="audiofile.mp3" type="audio/mpeg">
3   <source src="audiofile.ogg" type="audio/ogg">
4   Votre navigateur ne supporte pas l'élément <code>audio</code>.
5 </audio>

```

- controls** : Attribut qui ajoute des contrôles de lecture (play, pause, volume, etc.).
- <source>** : Permet de spécifier plusieurs formats de fichiers pour une meilleure compatibilité entre navigateurs. Le navigateur utilisera le premier format qu'il est capable de lire.
- src** : Chemin vers le fichier audio.
- type** : Type MIME du fichier audio.
- Fonctionnalités Avancées
- Lecture automatique** : **autoplay** (la lecture démarre automatiquement lorsque la page est chargée).
- Boucle** : **loop** (le fichier audio se répète indéfiniment).
- Préchargement** : **preload** (contrôle comment le fichier audio est préchargé ; valeurs possibles : **none**, **metadata**, **auto**).

B - BALISE <VIDEO>

La balise **<video>** est utilisée pour intégrer un contenu vidéo dans une page web.

- Syntaxe de Base

```

1 <video controls width="320" height="240">
2   <source src="movie.mp4" type="video/mp4">
3   <source src="movie.ogg" type="video/ogg">
4   Votre navigateur ne supporte pas l'élément <code>video</code>.
5 </video>
6

```

- controls** : Ajoute des contrôles de lecture vidéo.

- **width et height** : Définissent la largeur et la hauteur de la vidéo.
- **<source>** : Permet de spécifier plusieurs formats de fichiers vidéo.
- **src** : Chemin vers le fichier vidéo.
- **type** : Type MIME du fichier vidéo.
- Fonctionnalités Avancées
- **Lecture automatique** : **autoplay**.
- **Boucle** : **loop**.
- **Préchargement** : **preload**.
- **Muet** : **muted** (démarre la vidéo sans son).
- **Poster** : **poster** (image à afficher avant la lecture de la vidéo).

C - COMPATIBILITE ET FORMATS

La compatibilité des formats de fichiers audio et vidéo varie selon les navigateurs. Il est donc recommandé de fournir plusieurs formats de fichiers pour assurer une lecture optimale sur tous les navigateurs. Les formats courants pour l'audio incluent MP3 et Ogg, tandis que pour la vidéo, MP4, WebM et Ogg sont fréquemment utilisés.

En résumé, les balises **<audio>** et **<video>** simplifient grandement l'intégration de médias dans les pages web, offrant une compatibilité native étendue et des fonctionnalités de contrôle pour une expérience utilisateur améliorée.

2) LES PSEUDOS CLASSES

Les pseudo-classes en CSS sont des outils puissants pour cibler des éléments dans des états spécifiques ou des parties d'éléments. Les pseudo-classes comme **:active**, **:focus**, **:disabled**, et **:checked** permettent de modifier le style d'un élément en fonction de son état dans le DOM (Document Object Model).

A - :ACTIVE

La pseudo-classe **:active** est utilisée pour cibler un élément qui est en cours d'activation par l'utilisateur, typiquement lorsqu'un bouton ou un lien est en train d'être cliqué.

```
1 button:active {
2   color: red;
3 }
```

Dans cet exemple, le texte d'un bouton devient rouge lorsqu'il est activement cliqué.

B - :FOCUS

La pseudo-classe **:focus** cible un élément qui a reçu le focus. Cela peut se produire lorsque l'utilisateur clique sur un élément ou le sélectionne à l'aide du clavier (par exemple, en appuyant sur la touche Tab pour naviguer).

```
1 input:focus {
2   border-color: blue;
3 }
```

Ici, un champ de saisie (**input**) aura une bordure bleue lorsqu'il est en focus.

C - :DISABLED

La pseudo-classe **:disabled** cible les éléments de formulaire qui sont désactivés, ce qui signifie qu'ils ne sont pas interactifs et ne peuvent pas être sélectionnés.

```
1 button:disabled {
2   background-color: grey;
3   cursor: not-allowed;
4 }
```

Dans cet exemple, un bouton désactivé (**disabled**) aura un fond gris et un curseur indiquant qu'il n'est pas autorisé à être cliqué.

D - :CHECKED

La pseudo-classe **:checked** est spécifique aux éléments de formulaire comme les cases à cocher et les boutons radio. Elle permet de cibler ces éléments lorsqu'ils sont sélectionnés (cochés).

```
1 input[type="checkbox"]:checked {
2   background-color: green;
3 }
```

Cet exemple change la couleur de fond des cases à cocher lorsqu'elles sont cochées.

E - UTILISATION DANS LES FEUILLES DE STYLE

Ces pseudo-classes peuvent être utilisées pour améliorer l'expérience utilisateur en fournissant un retour visuel sur l'état des éléments interactifs. Par exemple, en changeant l'apparence des éléments lorsqu'ils sont en focus, vous pouvez rendre votre site plus accessible aux utilisateurs qui naviguent au clavier.

F - COMPATIBILITE

Ces pseudo-classes sont largement supportées par les navigateurs modernes, ce qui les rend fiables pour une utilisation dans la conception web responsive et accessible.

En résumé, **:active**, **:focus**, **:disabled**, et **:checked** sont des pseudo-classes CSS qui permettent de modifier le style des éléments en fonction de leur état, offrant ainsi une expérience utilisateur plus riche et interactive.

3) LES PSEUDOS ELEMENTS - CIBLAGE EN FONCTION DE LA POSITION

Les pseudo-éléments en CSS sont des sélecteurs spéciaux qui permettent de cibler des éléments spécifiques ou des parties d'éléments dans un groupe. Parmi eux, **:first-child**, **:last-child**, **:nth-child**, et **:nth-of-type** sont particulièrement utiles pour cibler des éléments en fonction de leur position dans un groupe de frères et sœurs (siblings) dans le DOM.

A - :FIRST-CHILD

La pseudo-classe **:first-child** cible le premier enfant d'un élément parent.

```
1 ul li:first-child {
2   color: red;
3 }
```

Dans cet exemple, le premier élément **li** dans chaque **ul** sera en rouge.

B - :LAST-CHILD

La pseudo-classe **:last-child** cible le dernier enfant d'un élément parent.

```
1 ul li:last-child {
2   color: blue;
3 }
```

Ici, le dernier élément **li** dans chaque **ul** sera en bleu.

C - :NTH-CHILD

La pseudo-classe **:nth-child** cible un ou plusieurs enfants spécifiques en fonction de leur position. Elle accepte un argument qui peut être un nombre, une formule, ou des mots-clés.

```
1 ul li:nth-child(2) {
2   color: green;
3 }
```

Cet exemple colore en vert le deuxième **li** de chaque **ul**.

Vous pouvez également utiliser des formules, comme **:nth-child(odd)** pour cibler tous les enfants impairs, ou **:nth-child(3n+1)** pour cibler chaque troisième enfant, en commençant par le premier.

D - :NTH-OF-TYPE

La pseudo-classe **:nth-of-type** est similaire à **:nth-child**, mais elle cible des éléments spécifiques en fonction de leur type (tag) et de leur position.

```

1 div p:nth-of-type(2) {
2   color: purple;
3 }
```

Dans cet exemple, le deuxième paragraphe (**p**) dans chaque **div** sera coloré en violet.

E - UTILISATION ET IMPORTANCE

Ces pseudo-classes sont extrêmement utiles pour appliquer des styles spécifiques à des éléments en fonction de leur position relative dans un groupe. Elles sont particulièrement pratiques dans les situations où vous ne pouvez pas ou ne voulez pas ajouter des classes ou des ID spécifiques à certains éléments. Par exemple, elles peuvent être utilisées pour styliser des listes, des tableaux, ou des grilles de manière dynamique sans avoir besoin de modifier le HTML.

COMPATIBILITE

Ces pseudo-classes sont bien supportées dans les navigateurs modernes, ce qui les rend fiables pour une utilisation dans divers projets web.

En résumé, **:first-child**, **:last-child**, **:nth-child**, et **:nth-of-type** offrent une méthode puissante et flexible pour cibler des éléments en fonction de leur position dans un groupe, permettant des mises en page complexes et des styles dynamiques avec un minimum de code supplémentaire.

4) LES PSEUDOS ELEMENTS - INSERTION DE CONTENU

Les pseudo-éléments **::before** et **::after** en CSS sont utilisés pour insérer du contenu avant ou après le contenu d'un élément sélectionné. Ils sont extrêmement utiles pour ajouter des ornements, des décorations ou des éléments de mise en forme sans avoir besoin de modifier le HTML. Ces pseudo-éléments sont souvent utilisés en conjonction avec la propriété **content**.

A - SYNTAXE DE BASE

```

1 element::before {
2   content: "quelque chose";
3   /* Autres styles */
4 }
5
6 element::after {
7   content: "autre chose";
8   /* Autres styles */
9 }
```

B - ::BEFORE

Le pseudo-élément **::before** crée un pseudo-élément qui est le premier enfant de l'élément ciblé. Il est souvent utilisé pour ajouter des icônes, des formes décoratives ou du texte avant le contenu d'un élément.

```

1 p::before {
2   content: "Note: ";
3   color: blue;
4 }

```

Dans cet exemple, chaque paragraphe (`p`) commencera par le mot “Note:” en bleu.

C - ::AFTER

Le pseudo-élément `::after` fonctionne de la même manière que `::before`, mais il insère le contenu après le contenu de l’élément ciblé.

```

1 p::after {
2   content: ".";
3   color: red;
4 }

```

Ici, un point rouge sera ajouté à la fin de chaque paragraphe.

D - UTILISATION DE CONTENT

- La propriété `content` est essentielle pour `::before` et `::after`. Sans elle, rien ne sera ajouté.
- `content` peut inclure du texte, des images (avec `url`), ou même être vide pour des besoins de style.
- Les valeurs spéciales comme `attr()` peuvent être utilisées pour insérer des données dynamiques.

E - APPLICATIONS COURANTES

1. **Décoration** : Ajouter des ornements autour des éléments.
2. **Icônes CSS** : Insérer des icônes avant ou après des liens, des titres, etc.
3. **Mise en Forme** : Créer des formes spéciales ou des effets visuels.
4. **Ajout de Contenu** : Insérer du contenu qui n'est pas directement dans le HTML (par exemple, des guillemets autour des citations).

F - POINTS IMPORTANTS

- `::before` et `::after` sont des pseudo-éléments, donc ils doivent être stylisés comme des éléments HTML. Ils peuvent avoir des propriétés comme `display`, `position`, `width`, `height`, etc.
- Ces pseudo-éléments ne peuvent pas être insérés dans des éléments auto-fermants (comme `img`, `br`, `input`, etc.), car ils n'ont pas de contenu interne.

G - EN BREF

`::before` et `::after` sont des outils puissants pour les développeurs front-end, permettant d'ajouter du contenu et des styles supplémentaires sans altérer le HTML. Ils offrent une flexibilité créative et peuvent être utilisés pour améliorer l'esthétique et la lisibilité du contenu web.

5) LES TRANSFORMATIONS

Les transformations CSS permettent de modifier l'apparence visuelle des éléments HTML en les déformant, les redimensionnant, les déplaçant ou les faisant pivoter. Ces transformations sont réalisées à l'aide de la propriété **transform** en CSS. Voici quelques-unes des transformations de base que vous pouvez appliquer :

A - TRANSLATE

translate() déplace un élément de sa position actuelle. Les valeurs sont généralement données en pixels (**px**) ou en pourcentages (**%**).

```
transform: translate(50px, 100px);
```

Cela déplace l'élément de 50 pixels vers la droite et de 100 pixels vers le bas.

B - SCALE

scale() modifie la taille d'un élément. Une valeur supérieure à 1 agrandit l'élément, tandis qu'une valeur inférieure à 1 le réduit.

```
transform: scale(2, 3);
```

Cela double la largeur de l'élément et triple sa hauteur.

C - ROTATE

rotate() fait pivoter un élément autour de son point central. La valeur est généralement donnée en degrés (**deg**).

```
transform: rotate(45deg);
```

Cela fait pivoter l'élément de 45 degrés dans le sens des aiguilles d'une montre.

D - SKEW

skew() incline un élément selon les axes X et Y. Les valeurs sont également données en degrés.

```
transform: skew(30deg, 20deg);
```

Cela incline l'élément de 30 degrés sur l'axe X et de 20 degrés sur l'axe Y.

E - COMBINAISON DE TRANSFORMATIONS

Vous pouvez combiner plusieurs fonctions de transformation en une seule propriété **transform**.

```
transform: rotate(30deg) scale(1.5);
```

Cet exemple fait pivoter l'élément de 30 degrés et augmente sa taille de 50%.

F - POINTS IMPORTANTS

- **Coordonnées** : Les transformations se font par rapport au centre de l'élément par défaut, mais vous pouvez changer ce point de référence avec la propriété **transform-origin**.
- **Performance** : Les transformations CSS sont généralement accélérées par le matériel, ce qui signifie qu'elles sont fluides et ne ralentissent pas les performances.
- **Animation** : Les transformations peuvent être animées ou combinées avec des transitions pour des effets dynamiques.

G - EXEMPLE D'UNE TRANSFORMATION

```

1  <!DOCTYPE html>
2  <html lang="fr">
3      <head>
4          <meta charset="UTF-8">
5          <title>Transformation</title>
6          <style>
7              div {
8                  width: 100px;
9                  height: 100px;
10                 background-color: red;
11                 transform: rotate(180deg) scale(1.2);
12             }
13         </style>
14     </head>
15     <body>
16         <h1>Transformation</h1>
17         <div></div>
18     </body>
19 </html>

```

Dans cet exemple, lorsque vous survolez le **div**, il pivote de 180 degrés et augmente sa taille de 20%.

H - EN BREF

Les transformations CSS offrent une manière puissante et flexible de modifier l'apparence des éléments HTML sans changer leur position dans le flux du document, permettant ainsi de créer des effets visuels intéressants et des interactions utilisateur dynamiques.

6) LES TRANSITIONS

Les transitions CSS permettent de créer des effets d'animation fluides lorsqu'une propriété CSS change de valeur. Elles sont souvent utilisées pour améliorer l'expérience utilisateur en rendant les interactions plus douces et plus naturelles.

A - SYNTAXE DE BASE

La transition en CSS se fait principalement via la propriété **transition**. Cette propriété est un raccourci pour plusieurs sous-propriétés :

1. **transition-property** : spécifie la ou les propriétés CSS à animer.
2. **transition-duration** : définit la durée de l'animation.
3. **transition-timing-function** : décrit comment la vitesse de l'animation change au fil du temps.
4. **transition-delay** : définit un délai avant le début de l'animation.

```

1 div {
2     transition: background-color 0.5s ease-in-out 0s;
3 }

```

Dans cet exemple, la couleur de fond (**background-color**) du **div** changera de manière animée sur une période de 0.5 secondes, avec un effet d'accélération puis de décélération (**ease-in-out**), et sans délai (**0s**).

B - TRANSITION SUR HOVER

Un usage courant des transitions est de changer l'apparence d'un élément lors du survol avec la souris.

```

1 button {
2   background-color: blue;
3   transition: background-color 0.3s ease;
4 }
5
6 button:hover {
7   background-color: green;
8 }
```

Ici, la couleur de fond d'un bouton passera de bleu à vert en 0.3 secondes lorsqu'il est survolé.

C - TRANSITION SUR PLUSIEURS PROPRIETES

Vous pouvez animer plusieurs propriétés en même temps.

```

1 div {
2   transition: opacity 0.3s, transform 0.3s;
3 }
4
5 div:hover {
6   opacity: 0.5;
7   transform: scale(1.1);
8 }
```

Dans cet exemple, à la fois l'opacité et la transformation de l'échelle (**scale**) du **div** sont animées lors du survol.

D - CONSEILS D'UTILISATION

- Performance** : Certaines propriétés, comme **transform** et **opacity**, sont plus performantes à animer que d'autres, comme **width** ou **margin**.
- Timing functions** : Utilisez des fonctions de timing (**ease**, **linear**, **ease-in**, **ease-out**, **ease-in-out**) pour contrôler le rythme de l'animation.
- Testez sur différents navigateurs** : Assurez-vous que vos transitions fonctionnent de manière fluide sur tous les navigateurs principaux.

E - EXEMPLE COMPLET D'UNE TRANSITION

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Transition</title>
6      <style>
7          div {
8              width: 100px;
9              height: 100px;
10             background-color: red;
11             transition: width 2s, background-color 2s, transform 2s;
12         }
13
14         div:hover {
15             width: 200px;
16             background-color: blue;
17             transform: rotate(180deg);
18         }
19     </style>
20 </head>
21 <body>
22     <h1>Transition</h1>
23     <div></div>
24 </body>
25 </html>
26

```

Dans cet exemple, lorsque vous survolez le **div**, sa largeur, sa couleur de fond et sa rotation changent de manière animée.

F - EN BREF

Les transitions CSS sont un moyen simple et puissant d'ajouter des animations fluides aux changements de propriétés CSS, améliorant ainsi l'expérience utilisateur et rendant l'interface plus interactive et engageante.

7) LES ANIMATIONS

Les animations CSS permettent de créer des effets visuels complexes et des animations sur des éléments HTML. Contrairement aux transitions, qui ne gèrent que le changement d'état entre deux états, les animations CSS peuvent inclure plusieurs états et configurations au sein d'une même séquence d'animation.

A - SYNTAXE DE BASE

Pour créer une animation en CSS, vous devez définir au moins deux choses : les keyframes de l'animation et la propriété **animation** sur l'élément que vous souhaitez animer.

B - KEYFRAMES

Les keyframes définissent les étapes de l'animation. Chaque keyframe décrit comment l'élément animé doit se rendre à un moment donné de l'animation.

```
1 @keyframes exemple {
2   0% { background-color: red; }
3   50% { background-color: yellow; }
4   100% { background-color: green; }
5 }
```

Dans cet exemple, l'animation commence avec un fond rouge, passe par le jaune à mi-chemin, et finit en vert.

C - APPLICATION DE L'ANIMATION

Ensuite, vous appliquez l'animation à un élément en utilisant la propriété **animation**.

```
1 div {
2   animation: exemple 2s infinite;
3 }
```

Ici, l'animation nommée **exemple** est appliquée à tous les **div**, avec une durée de 2 secondes et se répète indéfiniment (**infinite**).

D - PROPRIETES DE L'ANIMATION

- **animation-name** : le nom de l'animation, qui correspond aux keyframes définis.
- **animation-duration** : la durée totale de l'animation.
- **animation-timing-function** : le rythme de l'animation (par exemple, **linear**, **ease-in**, **ease-out**, **ease-in-out**).
- **animation-delay** : un délai avant le début de l'animation.
- **animation-iteration-count** : le nombre de fois où l'animation doit se répéter.
- **animation-direction** : la direction de l'animation (par exemple, **normal**, **reverse**, **alternate**).
- **animation-fill-mode** : définit comment un élément doit être stylisé avant le début ou après la fin de l'animation.

E - EXEMPLE D'UNE ANIMATION

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Animation</title>
6      <style>
7          @keyframes slide {
8              0% { transform: translateX(0); }
9              100% { transform: translateX(100px); }
10         }
11
12         div {
13             width: 100px;
14             height: 100px;
15             background-color: red;
16             position: relative;
17             animation: slide 3s infinite alternate;
18         }
19     </style>
20 </head>
21 <body>
22     <h1>Animation</h1>
23     <div></div>
24 </body>
25 </html>
26

```

Dans cet exemple, le **div** se déplace horizontalement de 100 pixels d'avant en arrière (**alternate**) de manière infinie.

F - CONSEILS D'UTILISATION

- **Performance** : Comme pour les transitions, certaines propriétés sont plus performantes à animer (comme **opacity** et **transform**).
- **Complexité** : Les animations CSS peuvent être plus complexes que les transitions, donc utilisez-les pour des effets plus élaborés.
- **Compatibilité** : Vérifiez la compatibilité des animations CSS sur différents navigateurs pour assurer une expérience utilisateur cohérente.

G - EN BREF

Les animations CSS offrent un moyen puissant de créer des animations dynamiques et interactives sur les pages web, permettant aux développeurs de créer des expériences utilisateur riches et engageantes sans avoir recours à JavaScript ou à des plugins externes.

8) LES VARIABLES CSS

L'utilisation des variables en CSS, officiellement connues sous le nom de "custom properties", permet une plus grande flexibilité et réutilisabilité dans la gestion des styles. Les variables CSS peuvent stocker des valeurs réutilisables telles que des couleurs, des tailles de police, ou des marges, ce qui facilite la maintenance et la cohérence du style sur de grands projets.

A - DEFINITION DES VARIABLES CSS

Les variables CSS sont définies en utilisant la syntaxe `--nom-de-la-variable`. Elles sont généralement définies dans le sélecteur `:root`, ce qui les rend accessibles globalement sur toute la page.

```
1 :root {  
2   --couleur-principale: #4CAF50;  
3   --espacement-standard: 10px;  
4 }
```

Dans cet exemple, deux variables sont définies : `--couleur-principale` pour une couleur, et `--espacement-standard` pour un espacement.

B - UTILISATION DES VARIABLES CSS

Pour utiliser une variable CSS, vous utilisez la fonction `var()`.

```
1 div {  
2   background-color: var(--couleur-principale);  
3   padding: var(--espacement-standard);  
4 }
```

Ici, le `div` utilisera la couleur et l'espacement définis dans les variables CSS.

C - AVANTAGES DES VARIABLES CSS

1. **Maintenance Facilitée** : Modifier une variable au même endroit met à jour tous les styles qui l'utilisent.
2. **Cohérence du Thème** : Facilite la gestion d'un thème cohérent sur un site web ou une application.
3. **Réutilisabilité** : Permet de réutiliser des valeurs dans différents styles sans répétition.

D - PORTEE DES VARIABLES

Les variables CSS respectent la cascade et l'héritage. Si une variable est définie dans un sélecteur spécifique, elle ne sera disponible que dans ce sélecteur et ses enfants.

```

1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Portée des variables</title>
6      <style>
7          :root {
8              --couleur-principale: #4CAF50;
9              --espace-ment-standard: 10px;
10         }
11
12         .container {
13             --couleur-texte: orange;
14         }
15
16     /* Utilisable seulement dans .container et ses
17     enfants */
18
19
20         div {
21             background-color: var(--couleur-principale);
22             padding: var(--espace-ment-standard);
23             /*
24                 ne fonctionne pas (car il n'est pas défini)
25                 color: var(--couleur-texte); :
26             */
27         }
28
29         .container {
30             color: var(--couleur-texte);
31         }
32     </style>
33 </head>
34 <body>
35     <h1>Portée des variables</h1>
36     <div>Un div stylisé avec des variables CSS.</div>
37     <div class="container">
38         <p>Un premier paragraphe.</p>
39         <p>Un second paragraphe.</p>
40     </div>
41 </body>
42 </html>
43

```

Dans cet exemple, la **div** ayant la classe **container** peut utiliser la variable CSS **--couleur-texte**.

E - COMPATIBILITE

Les variables CSS sont supportées dans la plupart des navigateurs modernes, mais il est toujours bon de vérifier leur compatibilité, surtout si vous ciblez des navigateurs plus anciens.

F - EN BREF

Les variables CSS offrent une méthode puissante et flexible pour gérer les styles, rendant le code plus propre, plus facile à maintenir, et plus cohérent. Elles sont particulièrement utiles dans les grands projets et les frameworks où la réutilisation et la maintenance des styles sont cruciales.

I - PREMIERS PAS	3
1) Structure de base d'une page web	3
2) La Balise <p> pour les paragraphes	3
3) La balise br	4
4) Les titres	5
5) Balise hr	6
6) Les hyperliens	6
a - Caractéristiques de la balise <a>	7
b - Attributs title et target	7
7) Les commentaires	7
8) Les images	8
9) Introduction au CSS	9
a - Grandes caractéristiques du CSS	10
b - Façons d'intégrer le CSS dans un document HTML	10
10) Class et Id	11
a - L'attribut class en HTML et CSS	11
b - Attribut id en HTML et CSS	11
c - Différences et choix entre class et id	11
d - En bref	12
11) Balises strong et em	12
a - Balise 	12
b - Balise 	12
c - Bonnes Pratiques	12
d - Conclusion	13
12) Balises div et span	13
a - Balise <div>	13
b - Balise 	13
c - Comment choisir ?	14
13) Le flux dans un document et la différence inline/block	14
a - Flux normal du document	14
b - Modification du flux avec display	14
c - Autres propriétés influant sur le flux	15
d - Conclusion	15
II - AMELIORATION 2	15
1) Le ciblage en CSS	15
A. Sélection par identifiant	15
B. Sélection par classe	15
C. Sélecteur descendant	16

D. Sélecteur enfant	16
E. Sélecteur de voisin direct	16
F. Sélection par nom d'élément	17
G - Conclusion	17
2) Ancre et lien vers un mail	17
a - Ancres dans une page HTML	17
b - Liens mailto	18
c - Conclusion	18
3) Représenter un contenu, <figure> et <figcaption>	19
a - Utilisation de <figure>	19
b - Utilisation de <figcaption>	19
c - Exemple d'utilisation de ces deux balises	19
d - Bonnes pratiques	19
e - En bref	20
4) Mise en forme du texte en CSS	20
5) Les listes en HTML et leur mise en forme en CSS	22
a - Listes non ordonnées ()	22
b - Listes ordonnées ()	22
c - Propriétés CSS pour la mise en forme des listes	22
d - Mise en œuvre	23
III - AMELIORATION 2	23
1) Utilisation de polices	23
a - Utilisation d'une Police du Poste de l'Utilisateur	23
b - Utilisation de @font-face pour des Polices Téléchargées	24
c - Utilisation d'une Google Font	24
d - Conclusion	24
2) Les dimensions d'un bloc et les unités	25
a - Utilisation de width et height	25
b - Considérations Importantes	25
c - Les unités en CSS	26
d- Comparaison et choix d'une unité	26
e - Exemple pratique de l'utilisation de width et height	27
3) Les espacements (padding et margin)	27
a - Padding (margin interne)	27
b - Margin (marge externe)	27
c - Box-sizing	28
D - Exemple pratique de ces trois propriétés	28
4) La gestion du dépassement avec overflow	28
a - Valeurs de la propriété overflow	28
b - Usages et applications	29
c - Exemple pratique	29
5) Bordure et arrondi	29
a - Propriété border	29

b - Propriété border-radius	30
D - Exemple Pratique	30
6) Les images de fond	30
a - Syntaxe de Base	31
b - Utilisation de Plusieurs Images	31
c - Contrôle de la répétition et du positionnement	31
d - Taille de l'image d'arrière-plan	31
e - Applications	32
7) Un formulaire de base	32
a - Structure de Base d'un Formulaire	32
b - Mise en œuvre dans un exemple	33
c - Points clefs	33
8) La balise label	33
a - Fonctionnalités essentielles	33
b - Deux exemples d'utilisation	34
c - Bonnes Pratiques	34
9) Les différents types d'input	34
10) Textarea et select	36
a - Balise <textarea>	36
b - Balise <select>	36
c - Utilisations	36
11) Fieldset et legend	37
a - Balise <fieldset>	37
b - Balise <legend>	37
c - Utilisations	38
12) Découverte des tableaux html - table, tr et td	38
a - Balise <table>	38
b - Balise <tr>	38
c - Balise <td>	38
d - Exemple utilisant ces trois balises	39
e - Utilisations	39
13) Structuration de tableau - thead, tbody et tfoot	39
a - Balise <th>	39
b - Balise <thead>	40
c - Balise <tbody>	40
d - Balise <tfoot>	40
e - Balise <caption>	41
f - Exemple Complet	41
14) Bordure de tableau - border et border-collapse	41
a - Propriété border	42
b - Propriété border-collapse	42
c - Exemple mettant en œuvre ces deux propriétés	42
15) Fusion de cellules - colspan et rowspan	43
a - Attribut colspan	43

b - Attribut rowspan	43
c - Combinaison de colspan et rowspan	43
d - Points essentiels	44
16) Les ombrages avec box-shadow	44
a - Syntaxe de box-shadow	44
b - Ombre Interne	45
Points clés	45
17) Les colonnes	45
a - Propriété column-count	45
b - Propriété column-gap	46
c - Propriétés column-rule-*	46
d - Exemple complet	46
e - Points clefs	47
18) Le survol avec :hover	47
a - Utilisation de :hover	47
Applications courantes de :hover	47
b - Points essentiels	48
19) Le masquage (display:none)	48
a - Fonctionnement de display: none;	48
b - Comparaison avec visibility: hidden;	49
c - Utilisations Courantes	49
d - Points Clés	49
IV - LA MISE EN PAGE	50
1) Le positionnement	50
a - static	50
b - relative	50
c - absolute	50
d - fixed	50
e - sticky	50
f - unset	50
g - Points essentiels	50
2) La position absolute	51
a - Fonctionnement de position: absolute;	51
b - Points essentiels	52
c - En bref	52
3) Position absolute et valeur négative	52
a - Fonctionnement	52
b - Deux exemples d'utilisation	52
c - Points essentiels	53
d - En bref	53
4) Position absolute et z-index	53
a - z-Index	53
b - Utilisation conjointe	54
d - Points essentiels	55

e - En bref	55
5) Position fixed	55
a - Fonctionnement de position: fixed;	55
b - Points essentiels	56
c - En bref	56
6) Position sticky	57
a - Fonctionnement de position: sticky;	57
b - Points essentiels	57
c - En bref	58
7) Inline-block, un comportement hybride	58
a - Principe de Fonctionnement	58
b - Utilisation	58
c - Points essentiels	59
d - En bref	59
8) Les tables avec display	60
a - Principe de fonctionnement	60
b - Usage	60
c - Points essentiels	61
d - En bref	62
9) Flexbox - vue d'ensemble	62
a - Principes de Base de Flexbox	62
b - Avantages de Flexbox	63
c - Limitations	64
d - En bref	64
10) Flexbox - gestion des axes	64
a - Flex direction	64
b - Justify content	64
c - Align items	64
d - Exemple utilisant ces propriétés	65
d - En bref	65
11) Flexbox - autres points importants	66
a - Flex wrap	66
b - Flex shrink	66
c - Flex grow	66
d - Exemple utilisant ces propriétés	67
e - En bref	68
12) Les grilles en CSS (grid layout)	68
a - Principes de base du grid layout	68
b - Avantages du grid layout	70
c - Limitation	70
d - En bref	70
13) Les media queries - principe général	70
a - Différence entre screen et print	70
b - Utilisation de plusieurs conditions	71
d - En bref	71

14) Les media queries - le mobile first	71
15) Flexbox et media queries	72
a - Exemple d'utilisation combinée	73
b - Avantages de cette approche	74
c - Conseils pour une meilleure pratique	74
d - En résumé	74
16) Grid layout et media queries	74
a - Exemple d'utilisation combinée	74
b - Avantages de cette approche	75
c - Conseils pour une meilleure pratique	75
V - PERFECTIONNEMENT	77
1) Les balises <audio> et <video>	77
a - Balise <audio>	77
b - Balise <video>	77
c - Compatibilité et formats	78
2) Les pseudos classes	78
a - :active	78
b - :focus	79
c - :disabled	79
d - :checked	79
e - Utilisation dans les feuilles de style	79
f - Compatibilité	79
3) Les pseudos éléments - ciblage en fonction de la position	80
a - :first-child	80
b - :last-child	80
c - :nth-child	80
d - :nth-of-type	80
e - Utilisation et importance	81
f - Compatibilité	81
4) Les pseudos éléments - insertion de contenu	81
a - Syntaxe de base	81
b - ::before	81
c - ::after	82
d - Utilisation de content	82
e - Applications courantes	82
f - Points Importants	82
g - En bref	82
5) Les transformations	82
a - Translate	83
b - Scale	83
c - Rotate	83
d - Skew	83
e - Combinaison de Transformations	83
f - Points importants	83
g - Exemple d'une transformation	84

h - En bref	84
6) Les transitions	84
a - Syntaxe de base	84
b - Transition sur hover	85
c - Transition sur plusieurs propriétés	85
d - Conseils d'utilisation	85
e - Exemple complet d'une transition	86
f - En bref	86
7) Les animations	86
a - Syntaxe de Base	86
d - Propriétés de l'Animation	87
e - Exemple d'une animation	88
f - Conseils d'Utilisation	88
g - En bref	88
8) Les variables CSS	88
a - Définition des Variables CSS	89
b - Utilisation des variables CSS	89
c - Avantages des variables CSS	89
d - Portée des variables	90
e - Compatibilité	90
f - En bref	91