CSS3
DU DÉBUTANT AU CONFIRMÉ

Auteur : Paterne Guélablé Gnonzion

Formateur développeur web et mobile : **PHP, Symfony, Rust ...**

Linkedin : Paterne G. G.

Github : teamflp

Procédure pour utiliser ce cours

Bienvenue à ce cours sur le CSS ! Pour tirer le meilleur parti de ce contenu et acquérir une compréhension solide de la matière, suivez les étapes recommandées ci-dessous.

1. **Préparation** :
    - Assurez-vous d'avoir les prérequis : connaissance basique du HTML.
    - Installez un bon éditeur de code (comme Visual Studio Code).
    - Configurez un navigateur moderne pour les tests (Chrome, Firefox, Safari, etc.).
2. **Étude systématique** :
    - Commencez par l'**introduction** pour comprendre le contexte du CSS.
    - Poursuivez avec les **bases** avant de passer aux concepts avancés.
    - Ne sautez pas de sections. Chaque partie est conçue pour s'appuyer sur la précédente.
3. **Pratiquez régulièrement** :
    - Après chaque section, prenez le temps de coder et de tester ce que vous avez appris.
    - Utilisez les exemples fournis et essayez de les modifier ou de les améliorer.
4. **Participez à des discussions** :
    - Rejoignez des forums ou des groupes dédiés au CSS pour poser des questions, partager vos connaissances ou aider d'autres apprenants.
    - Partager vos réalisations ou vos défis vous permettra d'obtenir des retours constructifs.
5. **Consultez des ressources supplémentaires** :
    - Explorez les **ressources et outils** mentionnés à la fin du cours pour approfondir vos connaissances et améliorer vos compétences.
6. **Mettez en pratique** :
    - Essayez de créer un petit projet web à partir de zéro, en appliquant tout ce que vous avez appris.

- Cela renforcera votre compréhension et vous donnera une idée des domaines dans lesquels vous pourriez avoir besoin de plus d'entraînement.

7. **Revenez régulièrement** :
   - La technologie et les normes évoluent. Revenez périodiquement pour mettre à jour vos connaissances ou rafraîchir certains concepts.
8. **Évaluez vos progrès** :
   - Testez régulièrement vos connaissances. Cela peut être à travers des quiz en ligne, des défis de codage ou des revues de code avec des pairs.

Bonne étude ! Rappelez-vous que l'apprentissage est un voyage, et chaque étape vous rapproche de la maîtrise du CSS.

        - `position: relative`
        - `position: absolute`
        - `position: fixed`
        - `position: sticky`
        - ## Flottants (`float`) et `clear` en CSS
        - `float`
        - `clear`
    - Couleurs et fonds
        - Types de couleurs : nommées, hexadécimales, rgb, rgba, hsl, hsla
        - Couleurs nommées
        - Couleurs hexadécimales
        - Couleurs RVB
        - Couleurs RVBA
        - Couleurs HSL
        - Couleurs HSLA
        - Background: couleur, image, position, taille, repeat, attachment, origin, clip, blend-mode, object-fit
        - `background-color`
        - `background-image`
        - `background-position`
        - `background-size`
        - `background-repeat`
        - `background-attachment`
        - `background-origin`
        - `background-clip`
        - `background-blend-mode`
        - `background`
        - `object-fit`
    - Typographie
        - Polices de caractères et @font-face
        - Poids, style et variante de fonte
        - Espacement des lettres, des mots et des lignes
        - Alignement du texte
    - Effets visuels
        - Ombres (box-shadow, text-shadow)
        - Bords arrondis (border-radius)
        - Transitions
        - Transformations
    - Unités et valeurs
        - Unités de longueur
        - Fonctions calc(), var() (variables CSS)
    - Sélecteurs Avancés en CSS
        - Sélecteurs d'attributs
        - Pseudo-classes
        - Pseudo-éléments
        - Combinateurs

Introduction to CSS

Le CSS, ou Cascading Style Sheets (Feuilles de style en cascade), est un langage de feuille de style utilisé pour décrire l'apparence et la mise en forme d'un document écrit en HTML ou en XML. Il s'agit de l'un des piliers essentiels du développement web, travaillant en tandem avec HTML pour créer des sites web visuellement attrayants et fonctionnellement robustes.

L'importance du CSS réside dans sa capacité à séparer le contenu (HTML) de sa présentation. Cette séparation offre plusieurs avantages :

- **Maintenabilité :** Le CSS permet de centraliser le style d'un site ou d'une application web, ce qui facilite les modifications globales de design sans avoir à retoucher individuellement chaque page.
- **Flexibilité :** Grâce aux sélecteurs et propriétés CSS, vous pouvez cibler spécifiquement des éléments pour les styliser, permettant une grande variété de designs.
- **Réactivité :** En utilisant le CSS, les développeurs peuvent créer des designs qui s'adaptent à différentes tailles d'écran, garantissant une expérience utilisateur optimale sur une variété de dispositifs, du desktop au mobile.
- **Performance :** En évitant le mélange de code HTML et de styles inline, le CSS peut être mis en cache, ce qui améliore les temps de chargement des sites.

Lorsqu'on débute en CSS, il est essentiel de comprendre certains concepts clés comme les sélecteurs, les propriétés, les valeurs et la spécificité. De plus, avec l'évolution du web, le CSS s'est également développé pour inclure des fonctionnalités plus avancées, telles que les animations, les transitions, Flexbox, Grid, et bien d'autres.

Aujourd'hui, avec l'ascension du mobile et la variété de dispositifs utilisés pour naviguer sur Internet, maîtriser le CSS est plus crucial que jamais pour tout développeur ou designer web souhaitant créer des sites ou des applications web modernes, attractives et fonctionnelles.

Qu'est-ce que le CSS?

Le **CSS** acronyme de `Cascading Style Sheets`, que l'on pourrait traduire par "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. Voici une explication détaillée :

Fonctionnement fondamental du CSS

Le CSS agit comme une série d'instructions indiquant à un navigateur web comment afficher un document. Ces instructions peuvent déterminer de nombreux aspects de la présentation, tels que la couleur du texte, la taille de la police, l'espacement entre les éléments, la mise en page des différentes sections d'une page, et bien plus encore. Séparation du contenu et de la présentation

L'une des principales philosophies du CSS est la séparation du contenu et de la présentation. Tandis que le HTML fournit la structure et le contenu d'un document (comme les titres, les paragraphes et les liens), le CSS définit son apparence et sa mise en forme. Cette séparation offre plusieurs avantages, notamment :

- Facilité de maintenance : Les styles peuvent être centralisés dans un ou plusieurs fichiers CSS externes, permettant des mises à jour cohérentes et centralisées du design d'un site web entier.
- Flexibilité : Le même contenu HTML peut être présenté de différentes manières en utilisant différentes feuilles de style. Par exemple, un site pourrait avoir un style pour l'affichage desktop et un autre pour l'affichage mobile.
- Accessibilité : Les utilisateurs peuvent appliquer leurs propres feuilles de style pour adapter les sites web à leurs besoins, comme augmenter la taille de la police pour une meilleure lisibilité.

La cascade

La "cascade" dans "Cascading Style Sheets" fait référence à l'ordre de priorité que le navigateur devrait suivre lorsqu'il décide quel style appliquer en cas de conflit. Cette cascade prend en compte la source des styles (par exemple, styles de l'utilisateur vs. styles du développeur) et la spécificité du sélecteur CSS utilisé. Évolution du CSS

Avec le temps, le CSS s'est énormément développé, offrant aux concepteurs et aux développeurs une vaste gamme d'options et de fonctionnalités, telles que les animations, les transitions, Flexbox, Grid, et les variables CSS, pour n'en nommer que quelques-unes.

En conclusion, le CSS est un outil puissant et essentiel pour quiconque souhaite créer des sites et applications web visuellement attrayants et fonctionnels. C'est le langage qui donne vie et style au contenu structuré par le HTML, permettant de créer des expériences web riches et interactives.

Comment intégrer du CSS : inline, interne, externe

Il y a trois manières principales d'intégrer du CSS dans une page web : inline, interne (ou embarqué) et externe. Chaque méthode a ses avantages et ses inconvénients. Voyons ces trois approches en détail :

1. **Inline CSS**

Le CSS inline est utilisé pour appliquer un `style` directement sur un élément individuel en utilisant l'attribut style dans la balise HTML.

Exemple :

```
<p style="color: blue; font-size: 16px;">Ceci est un paragraphe en bleu.
</p>
```

Avantages :

- Facile à mettre en œuvre
- Permet de remplacer les styles par défaut du navigateur
- Utile pour des styles spécifiques et uniques qui n'ont pas besoin d'être réutilisés.
- Garantit que le style s'applique à cet élément particulier, car il a une spécificité élevée.

Inconvénients :

- Augmente la taille du fichier HTML.
- Rend la maintenance difficile, surtout pour les grands sites.
- Mélange contenu (HTML) et présentation (CSS), ce qui va à l'encontre de la séparation recommandée du contenu et du style.

2. **CSS Interne (ou embarqué)**

Le CSS interne est défini dans l'en-tête du document HTML à l'aide de la balise `<style>`. Il s'applique à des éléments sur cette page spécifique.

Exemple :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <style>
      p {
        color: red;
        font-size: 18px;
      }
    </style>
  </head>
  <body>
    <p>Ceci est un paragraphe en rouge.</p>
  </body>
</html>
```

Avantages :

Avantages :

- Centralise les styles pour une page spécifique, ce qui peut faciliter certaines tâches de style.

Inconvénients :

- Si plusieurs pages utilisent le même style, vous devrez répéter le CSS sur chaque page.
- Augmente la taille du fichier HTML.

3. **CSS Externe**

Le CSS externe est défini dans un fichier séparé (généralement avec une extension `.css`). Ce fichier est ensuite lié à la page HTML à l'aide de la balise `<link>`. C'est la méthode recommandée pour la plupart des sites et applications web.

Exemple : Fichier `styles.css` :

```
p {
  color: green;
  font-size: 20px;
}
```

Dans le fichier HTML :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <link rel="stylesheet" href="styles.css" />
  </head>
  <body>
    <p>Ceci est un paragraphe en vert.</p>
  </body>
</html>
```

Avantages :

- Sépare complètement le contenu (`HTML`) de la présentation (`CSS`).
- Facilite la maintenance et la mise à jour des styles.
- Réduit la redondance car le même fichier CSS peut être utilisé sur plusieurs pages.
- Le navigateur peut mettre en cache le fichier CSS, accélérant le chargement des pages subséquentes.

Inconvénients :

- Une requête `HTTP` supplémentaire est nécessaire pour charger le fichier CSS externe, bien que cet impact soit généralement minime, voire inexistant, lorsque le fichier est mis en cache.

Recommandation : Pour de grands sites ou des applications, il est généralement recommandé d'utiliser le CSS externe en raison de ses avantages en matière de maintenance, de performance et de clarté.

Sélecteurs de base

Les sélecteurs CSS sont des motifs qui déterminent quels éléments de la page seront stylisés par les règles définies. Les sélecteurs de base sont les plus simples à comprendre et à utiliser. Voici une liste des sélecteurs de base avec une brève explication pour chacun :

1. **Sélecteur universel (*)**

Cible tous les éléments de la page.

Exemple :

```
* {
  margin: 0;
  padding: 0;
}
```

2. **Sélecteur de type ou sélecteur d'élément**

Cible un élément HTML spécifique par son nom de balise.

Exemple :

```
p {
  font-size: 16px;
}
```

3. **Sélecteur de classe (.)**

Cible un ou plusieurs éléments ayant une classe spécifique. Il est préfixé par un point.

Exemple : code HTML

```
Utilisation dans le HTML : <button class="btn">Cliquer ici</button>
```

Code CSS

```
.btn {
  background-color: blue;
  color: white;
}
```

Utilisation dans le HTML : Cliquer ici

4. **Sélecteur d'ID (#)**

Cible un élément spécifique ayant un attribut id particulier. Il est préfixé par un dièse #. Chaque ID doit être unique dans une page.

Exemple : HTML

```
<div id="header">...</div>
```

```
#header {
  background-color: gray;
}
```

5. **Sélecteur de groupe**

Cible plusieurs sélecteurs en même temps et leur applique le même ensemble de styles.

Exemple :

```
h1,
h2,
h3 {
  font-family: "Arial", sans-serif;
}
```

6. **Sélecteur de descendant**

Cible un élément qui est un descendant direct ou indirect d'un autre élément.

Exemple :

```
article p {
  color: green;
}
```

Ce code stylise tous les `<p>` qui sont à l'intérieur d'un élément `<article>`, qu'ils soient directs ou nichés plus profondément.

7. **Sélecteur d'enfant (>)**

Cible un élément qui est un enfant direct d'un autre élément.

Exemple :

```
article > p {
  color: red;
```

```
}
```

Ce code stylise uniquement les `<p>` qui sont des enfants directs de `<article>`, et non ceux qui sont plus profondément nichés.

8. **Sélecteur d'attribut**

Les sélecteurs d'attribut en CSS permettent de cibler des éléments en fonction de leurs attributs et des valeurs de ces attributs. Ils peuvent être très utiles pour appliquer des styles à des éléments qui ont des attributs spécifiques ou dont les valeurs d'attributs correspondent à certaines conditions.

Syntaxe de base

Pour cibler un élément en fonction de la présence d'un attribut spécifique, quelle que soit la valeur de cet attribut :

```
[element[attribute]] {
  /* styles */
}
```

Exemple : Cible tous les éléments `<a>` ayant un attribut href :

```
a[href] {
  color: blue;
}
```

- **Sélecteur d'attribut avec une valeur précise**

Pour cibler un élément dont l'attribut a une valeur précise :

```
[element[attribute="value"]] {
  /* styles */
}
```

Exemple : Cible tous les éléments `<a>` dont l'attribut href a la valeur `"#"`:

```
a[href="#"] {
  color: gray;
}
```

- **Autres opérateurs pour les sélecteurs d'attribut**

1. `^=` **(commence par) :** Cible les éléments dont la valeur de l'attribut commence par une chaîne spécifiée.

```
a[href^="https://"]
{
  font-weight: bold;
}
```

2. `$=` **(se termine par) :** Cible les éléments dont la valeur de l'attribut se termine par une chaîne spécifiée.

```
a[href$=".pdf"] {
  background: url("pdf-icon.png") no-repeat;
}
```

3. `*=` **(contient) :** Cible les éléments dont la valeur de l'attribut contient la chaîne spécifiée.

```
a[href*="example"] {
  color: red;
}
```

4. `~=` **(contient un mot) :** Cible les éléments dont la valeur de l'attribut contient un mot spécifié.

```
div[class~="warning"] {
  border: 1px solid red;
}
```

5. `|=` **(commence par un mot) :** Cible les éléments dont la valeur de l'attribut commence par un mot spécifié, suivi d'un tiret (`-`).

```
div[lang|="en"] {
  font-style: italic;
}
```

Les sélecteurs d'attribut peuvent offrir une grande flexibilité et précision lors de la stylisation des éléments. Ils peuvent être particulièrement utiles lorsque vous travaillez avec des éléments qui ont des attributs dynamiques ou lorsque vous souhaitez appliquer des styles basés sur des attributs spécifiques plutôt que sur des classes ou des ID.

Propriétés et valeurs

En CSS, les styles sont définis par des paires de propriétés et de valeurs. Ces paires sont utilisées pour spécifier comment un élément doit être présenté dans un navigateur.

- **Propriété :** C'est la caractéristique que vous souhaitez définir (par exemple, la couleur, la marge, la largeur, la fonte, etc.).
- **Valeur :** C'est la définition spécifique pour cette propriété (par exemple, "rouge", "20px", "Arial", etc.).

Exemple de propriété et valeur :

```
p {
  color: red;
  font-size: 16px;
}
```

- `color` est une propriété et red est sa valeur.
- `font-size` est une propriété et 16px est sa valeur.

Voici quelques propriétés couramment utilisées avec leurs possibles valeurs: color: Définit la couleur du texte. Valeurs possibles : red, #FF0000, rgb(255,0,0), etc.

Propriété	Définition	Valeurs possibles
`background-color`	couleur d'arrière-plan	yellow, #FFFF00 etc..
`font-family`	fonte à utiliser	Arial, 'Times New Roman', sans-serif, etc.
`font-size`	taille de la fonte	16px, 1em, 120%, etc.
`margin` et `padding`	marge extérieure et intérieure autour d'un élément	10px, 2em, 10px 5px 15px 5px, etc.
`border`	bordure autour d'un élément	1px solid black, 2px dashed red, etc.
`width` et `height`	largeur et la hauteur d'un élément	100px, 50%, auto, etc.
`display`	comment un élément doit être affiché	block, inline, flex, grid, none, etc.
`position`	type de positionnement d'un élément	static, relative, absolute, fixed, sticky, etc.

Il existe des centaines de propriétés CSS, chacune ayant son propre ensemble de valeurs possibles. Lorsque vous débutez en CSS, il est utile de se familiariser avec les propriétés les plus courantes et d'expérimenter pour voir comment elles affectent la présentation d'une page web. Avec le temps, vous serez plus à l'aise avec un plus grand nombre de propriétés et de valeurs et serez capable de styliser vos pages web avec précision et créativité.

La boîte modèle (model box)

Modèle de Boîte (Box Model) en CSS

Le modèle de boîte est un concept fondamental en CSS qui décrit comment les éléments sont structurés et dimensionnés sur une page web. Chaque élément est représenté comme une boîte rectangulaire, définie par plusieurs propriétés : marge, bordure, rembourrage et contenu.

Composants du Modèle de Boîte :

1. **Contenu (Content) :**
    - Zone où le texte, les images ou d'autres contenus sont affichés.
    - Les propriétés `width` et `height` déterminent sa taille.
2. **Rembourrage (Padding)**:
    - Espace entre le contenu et la bordure.
    - Propriétés : `padding-top`, `padding-right`, `padding-bottom`, `padding-left`.
3. **Bordure (Border)**:
    - Contour encadrant le rembourrage et le contenu.
    - Propriétés : `border-width`, `border-style`, `border-color`.
4. **Marge (Margin)**:
    - Espace extérieur à la boîte.
    - Propriétés : `margin-top`, `margin-right`, `margin-bottom`, `margin-left`.

Box Sizing

Comment la largeur et la hauteur d'un élément sont calculées.

- **content-box** (valeur par défaut) :
    - Largeur et hauteur concernent uniquement le contenu. Rembourrage et bordure s'ajoutent à ces dimensions.
- **border-box** :
    - Largeur et hauteur incluent le contenu, le rembourrage et la bordure (pas la marge).

Exemple :

```
.box {
  box-sizing: border-box;
  width: 300px;
  padding: 10px;
  border: 5px solid black;
}
```

Positionnement et mise en page

En CSS, le positionnement et la mise en page sont essentiels pour déterminer où et comment les éléments sont affichés sur la page. Plusieurs propriétés et concepts sont à la base de cette gestion.

Affichage (`display`) : block, inline, inline-block, none

La propriété `display` en CSS détermine comment un élément est traité en termes de modèle de boîte. C'est une des propriétés les plus fondamentales pour contrôler la mise en page des éléments sur une page web.

`display: block`

- **Description** : Les éléments avec `display: block` prennent toute la largeur disponible de leur conteneur parent et se positionnent sur une nouvelle ligne.
- **Caractéristiques** :
  - Crée une nouvelle ligne avant et après lui.
  - Accepte des propriétés de largeur (`width`) et de hauteur (`height`).
- **Exemples d'éléments** : `<div>`, `<p>`, `<h1>`, etc.

```
<div>Je suis un élément block</div>
```

```
div {
  display: block;
  background-color: lightblue;
}
```

`display: inline`

- **Description** : Les éléments avec `display: inline` ne prennent que l'espace nécessaire pour afficher leur contenu. Ils ne créent pas de nouvelle ligne avant ou après eux.
- **Caractéristiques** :
  - Ne crée pas de nouvelle ligne.
  - N'accepte pas des propriétés de largeur et de hauteur.
  - Les marges et les bordures peuvent ne pas fonctionner comme prévu.
- **Exemples d'éléments** : `<span>`, `<a>`, `<img>`, etc.

```
<span>Je suis</span> <span class="display">un élément inline</span>
```

```
span .display {
  display: inline;
  background-color: lightblue;
}
```

`display: inline-block`

- **Description** : Les éléments avec `display: inline-block` sont traités comme des éléments inline, mais ils acceptent des propriétés de dimensionnement comme `width` et `height`.
- **Caractéristiques** :
  - Ne crée pas de nouvelle ligne.
  - Accepte des propriétés de largeur et de hauteur.
  - Comportement hybride entre `inline` et `block`.
- **Utilisation** : Idéal pour des éléments qui doivent être alignés côte à côte mais qui ont également besoin de dimensionnement spécifique.

```
<span>Je suis un élément inline-block</span>
```

```
span {
  display: inline-block;
  width: 100px;
  height: 50px;
  background-color: lightgreen;
}
```

`display: none`

- **Description** : Les éléments avec `display: none` sont complètement retirés du flux de la page. Ils ne prennent aucun espace et ne sont pas visibles.
- **Caractéristiques** :
  - L'élément est totalement invisible.
  - N'affecte pas la mise en page des autres éléments.

```
<div>Vous ne me verrez pas sur la page.</div>
```

```
div {
  display: none;
}
```

Ce `<div>` ne sera pas visible et ne prendra pas de place sur la page.

Chaque valeur de la propriété display a ses propres caractéristiques et peut être utilisée pour atteindre différents effets de mise en page. En utilisant judicieusement ces valeurs, vous pouvez contrôler efficacement la disposition et le positionnement de vos éléments sur votre page web.

Positionnement : static, relative, absolute, fixed, sticky

La propriété `position` en CSS permet de définir le mode de positionnement d'un élément par rapport à son conteneur ou au viewport.

`position: static`

C'est le comportement par défaut d'un élément. L'élément est positionné selon l'ordre normal du flux de la page.

```
<div>Je suis un élément avec position static</div>
```

```
div {
  position: static;
}
```

`position: relative`

L'élément est positionné par rapport à sa position initiale, sans être retiré du flux normal.

```
<div>
  Je suis décalé de 10px à gauche et 5px vers le haut par rapport à ma
position
  initiale
</div>
```

```
div {
  position: relative;
  left: 10px;
  top: 5px;
}
```

`position: absolute`

L'élément est positionné par rapport au conteneur positionné le plus proche (autre que `static`) ou, à défaut, par rapport au viewport. Il est retiré du flux normal.

```
<div>
  Je suis positionné à 20px du haut et à droite de mon conteneur ou du
viewport
</div>
```

```

```

```
div {
  position: absolute;
  top: 20px;
  right: 0;
}
```

`position: fixed`

L'élément est positionné par rapport au viewport et ne bouge pas lors du défilement de la page.

```
<div>Je suis fixé en bas à gauche du viewport</div>
```

```
div {
  position: fixed;
  bottom: 10px;
  left: 10px;
}
```

`position: sticky`

L'élément est basé sur un mélange entre `relative` et `fixed`. Il "colle" à une position donnée lors du défilement.

```
<div>Je "colle" en haut lorsque vous faites défiler la page</div>
```

```
div {
  position: sticky;
  top: 0;
}
```

Le positionnement en CSS est essentiel pour créer des designs et des mises en page sophistiqués et interactifs. En maîtrisant ces valeurs, vous pourrez réaliser de nombreux effets sur votre site web.

Flottants (`float`) et `clear` en CSS

Les flottants sont utilisés en CSS pour déplacer un élément à gauche ou à droite, permettant ainsi aux autres contenus de s'enrouler autour de lui. C'est une technique qui a été largement utilisée pour la mise en page avant l'avènement de Flexbox et Grid, mais qui est toujours pertinente pour certains designs, notamment la mise en forme de textes autour des images.

`float`

La propriété `float` peut avoir l'une des valeurs suivantes :

- `left`: L'élément flotte à gauche.
- `right`: L'élément flotte à droite.
- `none`: L'élément ne flotte pas (valeur par défaut).

```
<img src="path/to/image.jpg" alt="Description de l'image" />
```

```
/* Faire flotter une image à gauche */
img {
  float: left;
}
```

`clear`

La propriété `clear` est utilisée pour contrôler le comportement des éléments par rapport aux éléments flottants précédents. Elle est utilisée pour empêcher que des éléments soient positionnés à côté d'éléments flottants précédents. Elle peut avoir l'une des valeurs suivantes :

- `left`: L'élément ne sera pas positionné à côté des éléments flottant à gauche.
- `right`: L'élément ne sera pas positionné à côté des éléments flottant à droite.
- `both`: L'élément ne sera pas positionné à côté de tout élément flottant (à gauche ou à droite).
- `none`: L'élément peut être positionné à côté des éléments flottants.

```
<!-- Flotter à gauche -->
<div class="float-left">Je flotte à gauche</div>

<!-- Empêcher le flottement à droite -->
<div class="clear-left">Je ne flotte pas à côté d'éléments à gauche</div>
```

```
/* Flotter à gauche */
.float-left {
  float: left;
}
```

Il est également courant d'utiliser un élément vide avec `clear: both;` après les éléments flottants pour s'assurer que le conteneur englobe correctement ses enfants flottants :

```
<!-- Flotter à gauche -->
<div class="float-left">Je flotte à gauche</div>

<!-- Flotter à droite -->
```

```
<div class="float-right;">Je flotte à droite</div>

<!-- Clear both pour s'assurer que le conteneur englobe ses enfants
flottants -->
<div class="clear-both;"></div>
```

```
/* Flotter à gauche */
.float-left {
  float: left;
}

/* Flotter à droite */
.float-right {
  float: right;
}

/* Clear both pour s'assurer que le conteneur englobe ses enfants flottants
*/
.clear-both {
  clear: both;
}
```

L'utilisation de `float` peut parfois entraîner des problèmes de débordement où le conteneur ne prend pas en compte la hauteur de ses enfants flottants. Dans ces cas, une technique courante consiste à utiliser un élément avec `clear: both` pour "nettoyer" après les éléments flottants.

NB : *Les flottants peuvent être délicats à gérer dans certaines situations, et des techniques telles que `Flexbox` ou `Grid` sont maintenant souvent privilégiées pour la mise en page. Cependant, comprendre comment fonctionnent les flottants reste important.*

Couleurs et fonds

Les couleurs et les fonds sont des éléments essentiels de la conception web. En CSS, il existe plusieurs façons de définir la couleur d'un élément, ainsi que de nombreuses propriétés pour contrôler les fonds.

Types de couleurs : nommées, hexadécimales, rgb, rgba, hsl, hsla

Il existe plusieurs façons de définir la couleur d'un élément en CSS. Chaque méthode a ses avantages et ses inconvénients, et il est important de comprendre les différences entre elles.

Couleurs nommées

Les couleurs nommées sont des mots-clés prédéfinis qui représentent une couleur spécifique. Elles sont faciles à utiliser et à retenir, mais elles sont limitées à un ensemble de couleurs prédéfinies.

Exemple :

```
p {
  color: red;
}
```

Couleurs hexadécimales

Les couleurs hexadécimales sont définies par une combinaison de 6 ou 3 valeurs hexadécimales (0-9, A-F) qui représentent les niveaux de rouge, vert et bleu (RVB) de la couleur. Elles sont très couramment utilisées en CSS.

Exemple :

```
p {
  color: #ff0000;
}
```

Couleurs RVB

Les couleurs RVB sont définies par une combinaison de 3 valeurs RVB (0-255) qui représentent les niveaux de rouge, vert et bleu de la couleur. Elles sont très couramment utilisées en CSS.

Exemple :

```
p {
  color: rgb(255, 0, 0);
}
```

Couleurs RVBA

Les couleurs RVBA sont définies par une combinaison de 4 valeurs RVB (0-255) qui représentent les niveaux de rouge, vert et bleu de la couleur, ainsi qu'une valeur alpha (0-1) qui représente la transparence de la couleur. Elles sont très couramment utilisées en CSS.

Exemple :

```
p {
  color: rgba(255, 0, 0, 0.5);
}
```

Couleurs HSL

Les couleurs HSL sont définies par une combinaison de 3 valeurs HSL (0-360, 0-100%, 0-100%) qui représentent la teinte, la saturation et la luminosité de la couleur. Elles sont très couramment utilisées en CSS.

Exemple :

```
p {
  color: hsl(0, 100%, 50%);
}
```

Couleurs HSLA

Les couleurs HSLA sont définies par une combinaison de 4 valeurs HSL (0-360, 0-100%, 0-100%) qui représentent la teinte, la saturation et la luminosité de la couleur, ainsi qu'une valeur alpha (0-1) qui représente la transparence de la couleur. Elles sont très couramment utilisées en CSS.

Exemple :

```
p {
  color: hsla(0, 100%, 50%, 0.5);
}
```

Background: couleur, image, position, taille, repeat, attachment, origin, clip, blend-mode, object-fit

La propriété `background` en CSS permet de définir la couleur et l'image d'arrière-plan d'un élément. Elle peut avoir plusieurs valeurs, séparées par des virgules, qui définissent différentes propriétés d'arrière-plan.

`background-color`

Définit la couleur d'arrière-plan d'un élément.

Les valeurs possibles sont :

- `color` : définit la couleur d'arrière-plan à partir d'un nom de couleur, d'une valeur hexadécimale, d'une valeur RVB, d'une valeur RVBA, d'une valeur HSL ou d'une valeur HSLA
- `transparent` : définit la couleur d'arrière-plan comme transparente
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur

Exemple :

```
p {
  background-color: red;
}
```

`background-image`

Définit l'image d'arrière-plan d'un élément.

Les valeurs possibles sont :

- `url("path/to/image.jpg")` : définit l'image d'arrière-plan à partir d'un chemin d'accès
- `none` : pas d'image d'arrière-plan
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur
- `linear-gradient()` : définit l'image d'arrière-plan à partir d'un dégradé linéaire
- `radial-gradient()` : définit l'image d'arrière-plan à partir d'un dégradé radial
- `repeating-linear-gradient()` : définit l'image d'arrière-plan à partir d'un dégradé linéaire répété
- `repeating-radial-gradient()` : définit l'image d'arrière-plan à partir d'un dégradé radial répété

Exemple :

```
p {
  background-image: url("path/to/image.jpg");
}
```

background-position

Définit la position d'arrière-plan d'un élément.

Les valeurs possibles sont :

- `left` : aligne l'image d'arrière-plan sur la gauche
- `right` : aligne l'image d'arrière-plan sur la droite
- `top` : aligne l'image d'arrière-plan en haut
- `bottom` : aligne l'image d'arrière-plan en bas
- `center` : aligne l'image d'arrière-plan au centre
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur
- `x% y%` : définit la position de l'image d'arrière-plan en pourcentage de la zone d'arrière-plan

Exemple :

```
p {
  background-position: top left;
}
```

background-size

Définit la taille d'arrière-plan d'un élément.

Les valeurs possibles sont :

- `auto` : taille par défaut
- `length` : définit la largeur et la hauteur de l'image d'arrière-plan en pixels
- `cover` : redimensionne l'image d'arrière-plan pour remplir la zone d'arrière-plan, en conservant le rapport hauteur/largeur et en coupant les bords
- `contain` : redimensionne l'image d'arrière-plan pour s'adapter à la zone d'arrière-plan, en conservant le rapport hauteur/largeur et en ajoutant des espaces vides
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur
- `percentage` : définit la largeur et la hauteur de l'image d'arrière-plan en pourcentage de la zone d'arrière-plan
- `auto length` : définit la largeur de l'image d'arrière-plan en pixels et la hauteur en pourcentage de la zone d'arrière-plan
- `length auto` : définit la largeur de l'image d'arrière-plan en pourcentage de la zone d'arrière-plan et la hauteur en pixels

Exemple :

```
p {
  background-size: 100px 50px;
}
```

background-repeat

Définit la répétition d'arrière-plan d'un élément.

Les valeurs possibles sont :

- `repeat` : l'arrière-plan est répété horizontalement et verticalement
- `repeat-x` : l'arrière-plan est répété horizontalement
- `repeat-y` : l'arrière-plan est répété verticalement
- `no-repeat` : l'arrière-plan n'est pas répété
- `space` : l'arrière-plan est répété horizontalement et verticalement, mais les images sont séparées par un espace
- `round` : l'arrière-plan est répété horizontalement et verticalement, mais les images sont redimensionnées pour s'adapter à l'espace
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur

Exemple :

```
p {
  background-repeat: no-repeat;
}
```

background-attachment

Définit l'attachement d'arrière-plan d'un élément.

Exemple :

```
p {
  background-attachment: fixed;
}
```

Les valeurs possibles sont :

- `scroll` : l'arrière-plan défiler avec le contenu
- `fixed` : l'arrière-plan reste fixe
- `local` : l'arrière-plan défiler avec le contenu
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur

background-origin

Définit l'origine d'arrière-plan d'un élément. Permet de décaler l'image de fond par rapport au contenu

Les valeurs possibles sont :

- `border-box` : l'arrière-plan est affiché dans la bordure
- `padding-box` : l'arrière-plan est affiché dans le padding
- `content-box` : l'arrière-plan est affiché dans le contenu
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur
- `text` : l'arrière-plan est affiché dans le texte
- `no-clip` : l'arrière-plan n'est pas affiché

Exemple :

```
/*  */
p {
  background-origin: content-box;
}
```

background-clip

Définit le clip d'arrière-plan d'un élément. Cette propriété permet de définir la zone d'affichage de l'arrière-plan.

Les valeurs possibles sont :

- `border-box` : l'arrière-plan est affiché dans la bordure
- `padding-box` : l'arrière-plan est affiché dans le padding
- `content-box` : l'arrière-plan est affiché dans le contenu
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur
- `text` : l'arrière-plan est affiché dans le texte
- `no-clip` : l'arrière-plan n'est pas affiché

Exemple :

```
p {
  background-clip: border-box;
}
```

background-blend-mode

Les modes de fusion possibles sont : Définit le mode de fusion d'arrière-plan d'un élément.

`normal`, `multiply`, `screen`, `overlay`, `darken`, `lighten`, `color-dodge`, `saturation`, `color`, `luminosity`, `difference`, `exclusion`, `hue`, `saturation`, `color`, `luminosity`, `initial`, `inherit`, `unset`.

Exemple :

```
p {
  background-blend-mode: multiply;
}
```

background

La propriété `background` est un raccourci. Elle peut également être utilisée pour définir plusieurs propriétés d'arrière-plan en même temps.

Exemple :

```
p {
  background: red url("path/to/image.jpg") no-repeat top left;
}
```

- red : couleur de fond
- url("path/to/image.jpg") : image de fond
- no-repeat : pas de répétition
- top : positionnement en haut
- left : positionnement à gauche

`object-fit`

La propriété `object-fit` en CSS permet de définir comment une image ou une vidéo doit s'adapter à son conteneur.

Les valeurs possibles sont :

- `fill` : l'image remplit le conteneur, en conservant ses proportions d'origine. Cela peut entraîner une déformation de l'image.
- `contain` : l'image s'adapte au conteneur, en conservant ses proportions d'origine. L'image peut ne pas remplir complètement le conteneur.
- `cover` : l'image remplit le conteneur, en conservant ses proportions d'origine. L'image peut être coupée.
- `none` : l'image conserve ses proportions d'origine et ne s'adapte pas au conteneur.

Exemple :

```
img {
  object-fit: cover;
}
```

Typographie

La typographie est un élément essentiel de la conception web. En CSS, il existe plusieurs propriétés pour contrôler la typographie d'un élément.

Polices de caractères et @font-face

Les polices de caractères sont un élément essentiel de la conception web. En CSS, il existe plusieurs façons de définir la police d'un élément.

Polices de caractères

- Les polices de caractères sont des fichiers de police qui sont chargés sur une page web. Elles peuvent être définies à l'aide de la règle `@font-face` en CSS.
- En plus des polices de caractères, il existe également des polices système qui sont installées sur l'ordinateur de l'utilisateur. Elles peuvent être utilisées en CSS en utilisant le nom de la police.
- En CSS, la propriété `font-family` est utilisée pour spécifier la police de caractère utilisée pour le texte.

```
/* Polices de caractères */
p {
  font-family: "Times New Roman", Times, serif;
}
```

`@font-face`

- La règle `@font-face` en CSS permet de charger des polices de caractères sur une page web.
- Elle peut être utilisée pour charger des polices de caractères personnalisées ou des polices de caractères système qui ne sont pas installées sur l'ordinateur de l'utilisateur.

```
/* Polices de caractères */
@font-face {
  font-family: "Times New Roman";
  src: url("path/to/font.woff2") format("woff2"), url("path/to/font.woff")
      format("woff");
}
```

Il est possible d'utiliser une librairie de polices de caractères comme Google Fonts pour charger des polices de caractères personnalisées sur une page web.

```
/* Polices de caractères */
@font-face {
  font-family: "Times New Roman";
  src: url("https://fonts.googleapis.com/css2?
family=Roboto:wght@100&display=swap");
}
```

Il est aussi possible d'importer la librairie de polices de caractères dans la balise `<head>`.

```
<link
  href="https://fonts.googleapis.com/css2?
family=Roboto:wght@100&display=swap"
  rel="stylesheet"
/>
```

Et ensuite de l'utiliser dans le CSS.

```
/* Polices de caractères */
p {
  font-family: "Roboto", sans-serif;
}
```

Poids, style et variante de fonte

Les propriétés CSS liées au poids, au style et à la variante de la fonte permettent de contrôler finement l'apparence du texte sur une page web.

- Poids : `font-weight`

Cette propriété définit l'épaisseur ou le poids de la fonte. Les valeurs courantes comprennent :

- `normal` : poids normal
- `bold` : texte en gras
- `bolder` : poids plus gras que le poids normal
- `lighter` : poids plus léger ou moins gras que le poids normal
- poids numérique (100, 200, 300, etc.) : Poids spécifiques pour les fontes offrant plusieurs poids.
- `initial` : valeur par défaut

```
<p>Je suis un texte en <span class="bold">gras</span></p>
```

```
.bold {
  font-weight: bold;
}
```

Résultat : Je suis un texte en **gras**

- Style : `font-style`

Cette propriété définit le style de la fonte. Les valeurs courantes comprennent :

- `normal` : style normal
- `italic` : texte italique
- `oblique` : texte incliné
- `initial` : valeur par défaut

```
<p>Je suis un texte en <span class="italic">italique</span></p>
```

```
.italic {
  font-style: italic;
}
```

Résultat : Je suis un texte en *italique*

- Variante : `font-variant`

La propriété `font-variant` permet de contrôler les variantes stylistiques d'une fonte. Elle a principalement été utilisée pour mettre en œuvre des petites majuscules, mais avec l'adoption de CSS3 et des spécifications de fontes, sa fonction s'est étendue pour couvrir d'autres variantes de fontes.

L'utilisation la plus courante de `font-variant` est pour les petites majuscules :

- **`normal`** : Aucune variante spéciale n'est appliquée.
- **`small-caps`** : Transforme les lettres minuscules en petites majuscules.
- **`all-small-caps`** : Transforme toutes les lettres en petites majuscules.

Les valeurs `small-caps` et `all-small-caps` sont souvent utilisées pour mettre en évidence des mots ou des phrases dans un texte.

```
<p>Je suis un texte en <span class="small-caps">petites majuscules</span>
</p>
```

```
.small-caps {
  font-variant: small-caps;
}
```

Résultat : Je suis un texte en PETITES MAJUSCULES

- Variante numérique

Avec l'évolution de CSS, d'autres sous-propriétés liées aux variantes numériques sont apparues, telles que :

- **`font-variant-numeric`** : permet de contrôler les variantes numériques d'une fonte.
- **`font-variant-east-asian`** : permet de contrôler les variantes asiatiques d'une fonte.
- **`font-variant-ligatures`** : permet de contrôler les ligatures d'une fonte.
- **`font-variant-caps`** : permet de contrôler les capitales d'une fonte.
- **`font-variant-alternates`** : permet de contrôler les variantes alternatives d'une fonte.
- **`font-variant-position`** : permet de contrôler la position des variantes d'une fonte.
- **`font-variant-emoji`** : permet de contrôler les variantes emoji d'une fonte.
- **`font-variant-language`** : permet de contrôler les variantes linguistiques d'une fonte.
- **`font-variant-numeric`** : permet de contrôler les variantes numériques d'une fonte.

Ces sous-propriétés permettent un contrôle plus précis sur différents aspects stylistiques des fontes, tels que la forme des chiffres, les ligatures, les variantes alternées, et plus encore.

Pour un contrôle maximal, il est souvent recommandé d'utiliser ces sous-propriétés plutôt que la propriété `font-variant` générale, car elles offrent une plus grande flexibilité et précision.

Note : L'utilisation effective de ces variantes dépend de la fonte en question. Si une fonte n'offre pas une variante stylistique spécifique, la spécification sera ignorée.

Espacement des lettres, des mots et des lignes

L'espacement entre les lettres, les mots et les lignes joue un rôle crucial dans la lisibilité et l'apparence du texte sur une page web.

`letter-spacing`

Cette propriété définit l'espace entre les caractères d'un texte.

- Valeur par défaut : `normal`
- Peut être défini en unités comme `em`, `px`, `pt`, etc.

Exemple :

```html
<p>Bonjour</p>
```

```css
p {
  letter-spacing: 20px;
}
```

Résultat :

B o n j o u r

`word-spacing`

Cette propriété définit l'espace entre les mots d'un texte.

- Valeur par défaut : `normal`
- Peut être défini en unités comme `em`, `px`, `pt`, etc.
- Les espaces supplémentaires entre les mots sont ajoutés à l'espace normal entre les mots.
- Les espaces négatifs sont autorisés.
- Les espaces supplémentaires ne sont pas ajoutés avant, après ou entre les mots vides.
- Les espaces supplémentaires ne sont pas ajoutés avant ou après les mots qui ont une propriété `white-space` de `pre`.
- Les espaces supplémentaires ne sont pas ajoutés avant ou après les mots qui ont une propriété `white-space` de `pre-wrap`.
- Les espaces supplémentaires ne sont pas ajoutés avant ou après les mots qui ont une propriété `white-space` de `pre-line`.
- Les espaces supplémentaires ne sont pas ajoutés avant ou après les mots qui ont une propriété `white-space` de `nowrap`.
- Les espaces supplémentaires ne sont pas ajoutés avant ou après les mots qui ont une propriété `white-space` de `break-spaces`.

Exemple :

```html
<p>Bonjour tout le monde</p>
```

```css
p {
  word-spacing: 20px;
}
```

Résultat :

Bonjour tout le monde

`line-height`

La propriété line-height détermine la hauteur de ligne du texte. Elle n'affecte pas seulement l'espacement vertical entre les lignes de texte, mais aussi la taille des boîtes de ligne dans lesquelles le texte est placé.

- Valeur par défaut : `normal`
- Peut être défini en unités comme `em`, `px`, `pt`, `%` etc.
- Les valeurs négatives sont autorisées.
- Les valeurs décimales sont autorisées.
- Les valeurs sans unité sont autorisées.
- Les valeurs de pourcentage sont autorisées.

Exemple :

```
p {
  line-height: 2;
}
```

Alignement du texte

L'alignement du texte est un élément essentiel de la conception web. En CSS, il existe plusieurs propriétés pour contrôler l'alignement du texte.

`text-align`

Cette propriété définit l'alignement horizontal du texte.

- Valeur par défaut : `left`
- Les valeurs possibles sont : `left`, `right`, `center`, `justify`, `initial`, `inherit`, `unset`.
- `justify` : aligne le texte sur les deux côtés, en ajoutant des espaces supplémentaires entre les mots.
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur
- `start` : aligne le texte à gauche
- `end` : aligne le texte à droite
- `match-parent` : aligne le texte sur le parent
- `justify-all` : aligne le texte sur les deux côtés, en ajoutant des espaces supplémentaires entre les mots.

Exemple :

```
<p>Je suis un texte aligné à <span class="right">droite</span></p>
```

```
.right {
  text-align: right; /* Le texte est aligné à droite */
}
```

Résultat :

Je suis un texte aligné à droite

`text-align-last`

Cette propriété définit l'alignement horizontal de la dernière ligne d'un texte.

- Valeur par défaut : `auto`
- Les valeurs possibles sont : `auto`, `left`, `right`, `center`, `justify`, `start`, `end`, `initial`, `inherit`, `unset`.
- `justify` : aligne le texte sur les deux côtés, en ajoutant des espaces supplémentaires entre les mots.
- `initial` : valeur par défaut
- `inherit` : hérite de la valeur de son parent
- `unset` : réinitialise la valeur
- `start` : aligne le texte à gauche
- `end` : aligne le texte à droite
- `match-parent` : aligne le texte sur le parent
- `justify-all` : aligne le texte sur les deux côtés, en ajoutant des espaces supplémentaires entre les mots.

Exemple :

```
<p>Je suis un texte aligné à <span class="right">droite</span></p>
```

```
.right {
  text-align-last: right; /* Le texte est aligné à droite */
}
```

Effets visuels

Les effets visuels en CSS sont essentiels pour améliorer l'esthétique d'un site web et fournir une expérience utilisateur dynamique. En CSS, il existe plusieurs propriétés pour contrôler les effets visuels d'un élément.

Ombres (box-shadow, text-shadow)

Permet d'ajouter une ombre à un élément. Vous pouvez spécifier des ombres intérieures et extérieures.

```
/* Syntaxe */
box-shadow: h-shadow v-shadow blur spread color inset;
```

- `h-shadow` : obligatoire. Définit la position horizontale de l'ombre. Une valeur positive place l'ombre à droite de la boîte, une valeur négative place l'ombre à gauche de la boîte.
- `v-shadow` : obligatoire. Définit la position verticale de l'ombre. Une valeur positive place l'ombre en dessous de la boîte, une valeur négative place l'ombre au-dessus de la boîte.
- `blur` : facultatif. Définit le flou de l'ombre. Plus la valeur est élevée, plus l'ombre est floue. La valeur par défaut est 0 (pas de flou).
- `spread` : facultatif. Définit la taille de l'ombre. Une valeur positive augmente la taille de l'ombre, une valeur négative la réduit. La valeur par défaut est 0 (l'ombre est de la même taille que l'élément).
- `color` : facultatif. Définit la couleur de l'ombre. La valeur par défaut est la couleur du texte.
- `inset` : facultatif. Définit si l'ombre est intérieure ou extérieure. La valeur par défaut est extérieure, `inset` spécifie que l'ombre est intérieure.

Exemple :

```
<div>Je suis un texte avec une ombre</div>
```

```
div {
  box-shadow: 10px 10px 5px grey;
}
```

- `10px` : position horizontale de l'ombre
- `10px` : position verticale de l'ombre
- `5px` : flou de l'ombre
- `grey` : couleur de l'ombre

Résultat :

Je suis un texte avec une ombre

`text-shadow` Applique une ombre au texte d'un élément.

```
<p>Je suis un texte avec une ombre</p>
```

```
p {
  text-shadow: 10px 10px 5px grey;
}
```

Résultat :

Je suis un texte avec une ombre

Bords arrondis (border-radius)

Permet de définir les bords arrondis d'un élément.

```
<div>Je suis un texte avec des bords arrondis</div>
```

```
div {
  border-radius: 25px;
}
```

Résultat :

Je suis un texte avec des bords arrondis

Transitions

Permettent de créer des changements fluides entre les valeurs des propriétés sur une durée donnée.

```
/* Syntaxe */
transition: property duration timing-function delay;
```

- `property` : obligatoire. Définit la propriété CSS à laquelle la transition doit s'appliquer.
- `duration` : obligatoire. Définit la durée de la transition.
- `timing-function` : facultatif. Définit la vitesse de transition.
- `delay` : facultatif. Définit le délai avant le début de la transition.
- `initial` : valeur par défaut

Exemple :

```
<div>Je suis un texte avec une transition</div>
```

```
div {
  transition: background-color 2s;
}
div:hover {
  background-color: red;
}
```

Transformations

Les transformations permettent de modifier la position, l'échelle, et d'autres aspects d'un élément.

Les propriétés de transformation sont :

- `transform` : définit une transformation 2D ou 3D sur un élément.
- `transform-origin` : définit le point d'origine d'une transformation 2D ou 3D.
- `transform-style` : définit si les enfants d'un élément doivent être positionnés dans l'espace 2D ou 3D.
- `perspective` : définit la perspective d'une transformation 3D.
- `perspective-origin` : définit le point d'origine de la perspective d'une transformation 3D.
- `backface-visibility` : définit si le visage arrière d'un élément doit être visible lorsqu'il est tourné vers l'utilisateur.
- `translate()` : définit une transformation 2D en spécifiant une valeur pour les coordonnées X et Y.
- `translateX()` : définit une transformation 2D en spécifiant une valeur pour la coordonnée X.
- `translateY()` : définit une transformation 2D en spécifiant une valeur pour la coordonnée Y.
- `translateZ()` : définit une transformation 3D en spécifiant une valeur pour la coordonnée Z.
- `scale()` : définit une transformation 2D en spécifiant une valeur pour les coordonnées X et Y.
- `scaleX()` : définit une transformation 2D en spécifiant une valeur pour la coordonnée X.
- `scaleY()` : définit une transformation 2D en spécifiant une valeur pour la coordonnée Y.
- `scaleZ()` : définit une transformation 3D en spécifiant une valeur pour la coordonnée Z.
- `rotate()` : définit une transformation 2D en spécifiant une valeur d'angle.
- `rotateX()` : définit une transformation 3D en spécifiant une valeur d'angle pour l'axe X.
- `rotateY()` : définit une transformation 3D en spécifiant une valeur d'angle pour l'axe Y.
- `rotateZ()` : définit une transformation 3D en spécifiant une valeur d'angle pour l'axe Z.
- `skew()` : définit une transformation 2D en spécifiant une valeur pour les coordonnées X et Y.
- `skewX()` : définit une transformation 2D en spécifiant une valeur pour la coordonnée X.
- `skewY()` : définit une transformation 2D en spécifiant une valeur pour la coordonnée Y.
- `matrix()` : définit une transformation 2D en spécifiant les valeurs d'une matrice 2D à 6 paramètres.
- `matrixX()` : définit une transformation 2D en spécifiant
- `matrixY()` : définit une transformation 2D en spécifiant
- `matrixZ()` : définit une transformation 3D en spécifiant
- `matrix3d()` : définit une transformation 3D en spécifiant les valeurs d'une matrice 3D à 16 paramètres.
- `translate3d()` : définit une transformation 3D en spécifiant une valeur pour les coordonnées X, Y et Z.
- `scale3d()` : définit une transformation 3D en spécifiant une valeur pour les coordonnées X, Y et Z.
- `rotate3d()` : définit une transformation 3D en spécifiant une valeur d'angle pour les coordonnées X, Y et Z.

Exemple :

```
<div>Je suis un texte avec une transformation</div>
```

```
div {
  transform: rotate(20deg);
}
```

Résultat :

- Les transformations 2D et 3D peuvent être combinées.

Exemple :

```
<div>Je suis un texte avec une transformation</div>
```

```
div {
  transform: rotate(20deg) translate(50px, 50px);
}
```

Résultat :

- Les transformations peuvent être animées en utilisant la propriété `transition`.

Exemple :

```
<div>Je suis un texte avec une transformation</div>
```

```
div {
  transition: transform 2s;
}
div:hover {
  transform: rotate(20deg) translate(50px, 50px);
}
```

Résultat :

Les transformations peuvent être animées en utilisant la propriété `animation` et `@keyframes`.

- La règle `@keyframes` définit les étapes ou "frames" d'une séquence d'animation. En spécifiant des pourcentages, vous définissez l'état de l'animation à différents points dans le temps.
- La propriété `animation` est utilisée pour appliquer une animation à un élément. Vous pouvez spécifier la durée, le timing, le nombre de répétitions, et bien d'autres paramètres.

Exemple :

```
<div>Je suis un texte avec une transformation</div>
```

```
div {
  animation: rotate 3s linear infinite;
}

@keyframes rotate {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Dans l'exemple ci-dessus, un élément

avec la classe animated sera animé en effectuant une rotation de 360 degrés sur une durée de 3 secondes, et cela de manière infinie.

Il est important de noter que pour obtenir une animation fluide et continue, la valeur de départ et la valeur finale de la transformation dans `@keyframes` doivent être cohérentes. Dans l'exemple donné, l'animation commence à 0 degré et se termine à 360 degrés, créant ainsi une rotation complète.

L'utilisation des animations CSS pour animer les transformations offre de nombreuses possibilités créatives pour améliorer l'expérience utilisateur sur un site web.

Unités et valeurs

Les unités et valeurs en CSS sont essentielles pour spécifier la taille, l'espacement, et d'autres aspects stylistiques des éléments

Unités de longueur

Les unités de longueur sont utilisées pour spécifier la taille d'un élément. Les unités de longueur les plus courantes sont :

- `px` : pixels
- `em` : taille de la police de l'élément
- `rem` : taille de la police de l'élément racine
- `vw` : 1% de la largeur de la fenêtre
- `vh` : 1% de la hauteur de la fenêtre
- `vmin` : 1% de la largeur ou de la hauteur de la fenêtre
- `vmax` : 1% de la largeur ou de la hauteur de la fenêtre
- `%` : pourcentage de la taille de l'élément parent

`px` : pixels

Une unité fixe qui représente un point à l'écran. Par exemple, `font-size: 16px;` définit une taille de police de 16 pixels.

Exemple :

```
p {
  font-size: 16px;
}
```

`em` : taille de la police de l'élément

Une unité relative à la taille de police de l'élément parent. Si l'élément parent a une `font-size` de `16px`, alors `1em` équivaut à `16px`.

Exemple :

```
p {
  font-size: 1em;
}
```

`rem` : taille de la police de l'élément racine

Similaire à `em`, mais au lieu de se baser sur la taille de police de l'élément parent, elle se base sur la taille de police de l'élément racine (`<html>`).

Exemple :

```
p {
  font-size: 1rem;
}
```

`vw` et `vh` : 1% de la largeur de la fenêtre

`vw` et `vh` sont des unités relatives à la taille de la fenêtre. `1vw` équivaut à `1%` de la largeur de la fenêtre, et `1vh` équivaut à `1%` de la hauteur de la fenêtre.

Exemple :

```
p {
  font-size: 1vw;
}

p {
  font-size: 1vh;
}
```

`vmin` et `vmax` : 1% de la largeur ou de la hauteur de la fenêtre

`vmin` et `vmax` sont des unités relatives à la taille de la fenêtre. `1vmin` équivaut à `1%` de la largeur ou de la hauteur de la fenêtre, et `1vmax` équivaut à `1%` de la largeur ou de la hauteur de la fenêtre, selon la valeur la plus petite ou la plus grande.

Exemple :

```
p {
  font-size: 1vmin;
}

p {
  font-size: 1vmax;
}
```

`%` : pourcentage de la taille de l'élément parent

Une unité relative à la taille de l'élément parent. `100%` équivaut à la taille de l'élément parent.

Exemple :

```
p {
  font-size: 100%;
}
```

Fonctions calc(), var() (variables CSS)

En CSS, les fonctions `calc()` et `var()` sont utilisées pour effectuer des calculs et définir des variables.

`calc()`

La fonction `calc()` est utilisée pour effectuer des calculs sur les valeurs CSS. Elle peut être utilisée pour effectuer des calculs sur les valeurs de longueur, les valeurs numériques, et même les valeurs de temps.

Exemple :

```
p {
  width: calc(100% - 50px);
}
```

Explication :

La propriété `width` de l'élément `<p>` est définie à `calc(100% - 50px)`. Cela signifie que la largeur de l'élément `<p>` sera égale à `100%` de la largeur de son parent, moins `50px`.

`var()`

Utilisée pour accéder aux valeurs des variables CSS, aussi appelées propriétés personnalisées. Par exemple, si vous avez défini `--main-color: #ff5733;`, vous pouvez utiliser cette variable comme ceci : `color: var(--main-color);`.

Exemple :

```
/* Définition de la variable */

:root {
  --main-color: #ff5733;
}

p {
  color: var(--main-color);
}
```

Résultat :

Je suis un texte avec une couleur rouge.

Explication :

La propriété `color` de l'élément `<p>` est définie à `var(--main-color)`. Cela signifie que la couleur du texte de l'élément `<p>` sera égale à la valeur de la variable `--main-color`.

La définition des variables CSS se met dans le sélecteur `:root` ou `html`.

L'utilisation de ces unités et fonctions permet une grande flexibilité dans la conception des styles CSS, en adaptant le design aux différentes tailles d'écran et en favorisant la réutilisabilité du code.

Sélecteurs Avancés en CSS

Les sélecteurs avancés en CSS offrent une précision accrue dans le ciblage des éléments, permettant des styles plus complexes et nuancés

Sélecteurs d'attributs

Les sélecteurs d'attributs permettent de cibler des éléments en fonction de leurs attributs et valeurs.

- **[attribut]** : Sélectionne les éléments avec cet attribut.

```
input[type="text"] {
  border: 1px solid #ddd;
}
```

- **[attribut="valeur"]** : Sélectionne les éléments avec cet attribut et cette valeur exacte.

```
input[type="submit"] {
  background-color: blue;
}
```

Pseudo-classes

Les pseudo-classes permettent de cibler des éléments en fonction de leur état ou de leur position.

- **:hover** : Cible un élément lorsqu'il est survolé.

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

- **:nth-child(n)** : Cible le n-ème enfant d'un élément.

```
li:nth-child(odd) {
  background-color: #f2f2f2;
}
```

Pseudo-éléments

Les pseudo-éléments permettent de cibler une partie spécifique d'un élément.

- **::before** : Utilisé pour insérer du contenu avant le contenu de l'élément.

```
p::before {
  content: "Intro : ";
}
```

- **::after** : Utilisé pour insérer du contenu après le contenu de l'élément.

```
p::after {
  content: " - Fin";
}
```

Combinateurs

Les combinateurs permettent de définir des relations spécifiques entre les sélecteurs.

- **Espace ( )** : Cible les descendants.

```
article p {
  color: gray;
}
```

- **>** : Cible les enfants directs.

```
ul > li {
  list-style-type: square;
}
```

- **+** : Cible l'élément immédiatement suivant.

```
h2 + p {
  font-size: 1.2em;
}
```

En maîtrisant ces sélecteurs avancés, vous pouvez créer des feuilles de style CSS beaucoup plus puissantes et spécifiques, offrant une grande flexibilité dans la conception de votre site web.

Responsivité et Media Queries

La création de designs web adaptatifs est essentielle à l'ère actuelle, où les utilisateurs accèdent à du contenu sur une multitude de dispositifs différents. Les Media Queries et les points d'arrêt sont des outils cruciaux pour réaliser cette adaptabilité.

Responsivité et Media Queries

Le web d'aujourd'hui est consulté depuis une variété d'appareils : ordinateurs de bureau, tablettes, smartphones, et bien d'autres. La conception adaptative, ou "responsive design", permet de s'assurer que votre site ou application web offre une expérience utilisateur optimale sur tous ces appareils.

Créer un design adaptatif

Le design adaptatif repose sur le principe de flexibilité. Cela signifie que votre conception doit s'adapter et se redimensionner en fonction des caractéristiques de l'appareil utilisé.

- **Grille fluide** : Utilisez des unités relatives, comme les pourcentages, plutôt que des pixels fixes pour les éléments de mise en page.
- **Images flexibles** : Assurez-vous que les images peuvent se redimensionner dans les conteneurs qui les entourent.
- **Media Queries** : Adaptez le style et la mise en page en fonction des capacités de l'appareil.

Utilisation des points d'arrêt (breakpoints)

Les "breakpoints" ou points d'arrêt, définis avec les Media Queries, permettent d'appliquer différents styles CSS en fonction de la taille de l'écran ou d'autres caractéristiques de l'appareil.

Les points d'arrêt :

```
/* Style par défaut pour les appareils plus petits (mobiles) */
body {
  font-size: 16px;
}

/* Style pour mobile */
@media (min-width: 320px) {
  body {
    font-size: 16px;
  }
}

/* Style pour les tablettes (paysage) */
@media (min-width: 768px) {
  body {
    font-size: 18px;
  }
}

/* Style pour les ordinateurs de bureau */
@media (min-width: 1024px) {
  body {
    font-size: 20px;
  }
}
```

```
/* Style pour les grands Écrans (Large Desktop) */
@media (min-width: 1200px) {
  body {
    font-size: 24px;
    /* Autres styles pour grand écran... */
  }
}

/* Style pour les imprimantes */
@media print {
  body {
    font-size: 12px;
    /* Autres styles pour impression... */
  }
}

/* Style pour les écrans haute résolution */
@media (-webkit-min-device-pixel-ratio: 2), (min-resolution: 192dpi) {
  body {
    font-size: 24px;
    /* Autres styles pour écrans haute résolution... */
  }
}

/* Style pour les écrans basse résolution */
@media (-webkit-max-device-pixel-ratio: 1.5), (max-resolution: 144dpi) {
  body {
    font-size: 16px;
    /* Autres styles pour écrans basse résolution... */
  }
}
```

Media Query pour les Mobiles et Tablettes en Mode Portrait

Pour cibler des appareils ayant une largeur d'écran comprise entre 320 pixels (généralement la largeur minimale d'un téléphone mobile en mode portrait) et 767 pixels (juste en dessous de la largeur couramment utilisée pour les tablettes en mode paysage), vous pouvez utiliser la media query suivante :

```
/* Style par défaut pour les appareils plus petits (mobiles) */
body {
  font-size: 16px;
}

@media (min-width: 320px) and (max-width: 767px) {
  body {
    font-size: 16px;
  }
}
```

Cette media query garantit que le font-size du body sera réglé sur 16px pour les appareils qui correspondent à cette plage de largeur d'écran.

Media Query pour les Tablettes en Mode Paysage

Pour cibler des appareils ayant une largeur d'écran comprise entre 768 pixels (généralement la largeur minimale d'une tablette en mode paysage) et 1023 pixels (juste en dessous de la largeur couramment utilisée pour les ordinateurs de bureau), vous pouvez utiliser la media query suivante :

Il est toujours recommandé de tester le design sur différents appareils pour s'assurer qu'il offre une expérience utilisateur optimale.

Préprocesseurs CSS

Les préprocesseurs CSS offrent une manière d'écrire des styles de manière plus concise, organisée et souvent plus compréhensible. Ils permettent aux développeurs d'utiliser des fonctionnalités de programmation non disponibles en CSS natif et compilent ensuite ce code en CSS standard que les navigateurs peuvent interpréter.

Pourquoi utiliser un préprocesseur ?

1. **Variables**: Stockez des valeurs fréquemment utilisées (comme des couleurs ou des espacements) et réutilisez-les dans tout le fichier, pour une maintenance plus facile.
2. **Imbrication**: Organisez vos styles de manière plus logique en imbricant les sélecteurs les uns dans les autres.
3. **Mixins**: Créez des ensembles réutilisables de styles.
4. **Héritage**: Partagez un ensemble commun de propriétés entre plusieurs sélecteurs.
5. **Opérations**: Réalisez des opérations mathématiques pour calculer des valeurs.
6. **Fonctions**: Utilisez des fonctions pour manipuler des valeurs, par exemple pour changer la luminosité d'une couleur.

Préprocesseurs populaires

Voici quelques-uns des préprocesseurs CSS les plus populaires :

- **Sass (et SCSS)**: Un préprocesseur riche et stable, largement adopté dans l'industrie.
- **Less**: Similaire à Sass, mais avec quelques différences dans la syntaxe et les fonctionnalités.
- **Stylus**: Offre une syntaxe très concise et flexible.
- **PostCSS**: Un post-processeur qui vous permet d'utiliser des plugins pour transformer votre CSS.

Exemple avec Sass

Voici un exemple simple de ce que vous pouvez faire avec Sass :

```
$primary-color: #3498db;

.button {
  background-color: $primary-color;
  &:hover {
    background-color: darken($primary-color, 10%);
```

```
  }
}
```

Ce code Sass serait ensuite compilé en CSS standard ressemblant à ceci :

```
.button {
  background-color: #3498db;
}

.button:hover {
  background-color: #2980b9;
}
```

Pour commencer à utiliser un préprocesseur, vous devrez généralement installer un outil de compilation pour transformer le code du préprocesseur en CSS standard. Cela peut être fait à l'aide d'outils tels que `Webpack`, `Gulp`, ou même des extensions de navigateur dédiées.

Pour apprendre à utiliser Sass, consultez la documentation officielle.

Méthodologies CSS

Les méthodologies CSS ont été développées pour aider les développeurs à écrire des codes CSS plus organisés, maintenables et compréhensibles. Chaque méthodologie a ses propres principes et recommandations.

Méthodologies CSS

Écrire du CSS pour de petites applications ou sites web est souvent simple, mais à mesure que le projet grandit, le CSS peut rapidement devenir désordonné, répétitif et difficile à maintenir. Les méthodologies CSS ont été créées pour résoudre ces problèmes, en offrant des lignes directrices sur la manière d'écrire du CSS de manière efficace et organisée.

BEM (Block Element Modifier)

- Principes : BEM est une méthodologie qui recommande une convention de nommage pour les classes CSS, ce qui facilite la compréhension de la relation entre le HTML et le CSS.
- **Syntaxe**: `.block__element--modifier`
  - **Block**: Représente l'élément principal.
  - **Element**: Représente un descendant du block.
  - **Modifier**: Indique une variante ou un état.
- **Exemple**:

```
.button {
} /* Block */
.button__icon {
```

```
} /* Element */
.button--large {
} /* Modifier */
```

OOCSS (Object Oriented CSS)

- Principes: OOCSS divise le code en deux catégories principales : structure et skin. Cela permet de réutiliser le code et de maintenir des styles plus cohérents.

- **Structure**: Définit la structure de base des éléments, comme les grilles.
- **Skin**: Définit l'apparence visuelle des éléments, comme les couleurs.
- **Exemple**:

```
.button {
} /* Structure */
.button--red {
} /* Skin */
```

SMACSS (Scalable and Modular Architecture for CSS)

- Principes: SMACSS divise le design en cinq catégories. Cela encourage la modularité et la scalabilité :

- **Base**: Définit les styles de base pour les éléments HTML.
- **Layout**: Définit la structure de base de la page.
- **Module**: Définit les modules réutilisables.
- **State**: Définit les états spécifiques des modules.
- **Theme**: Définit les styles spécifiques au thème.
- **Exemple**:

```
.button {
} /* Base */
.button--large {
} /* Layout */
.button__icon {
} /* Module */
.button.is-active {
} /* State */
.button--red {
} /* Theme */
```

L'adoption de ces méthodologies n'est pas mutuellement exclusive ; vous pouvez combiner des éléments de chacune pour créer un système qui fonctionne le mieux pour votre projet. L'important est d'assurer une architecture CSS cohérente, maintenable et compréhensible.

Astuces avancées et optimisation

Alors que vous devenez de plus en plus à l'aise avec le CSS, vous découvrirez certaines subtilités et techniques avancées qui peuvent améliorer la qualité de votre code et les performances de votre site.

z-index et empilement contextuel

Le `z-index` permet de contrôler l'ordre d'empilement des éléments. Cependant, son comportement peut parfois être déroutant.

- Un élément avec un `position` différent de `static` crée un nouveau contexte d'empilement.
- `z-index` n'affecte un élément que si cet élément a une valeur `position` autre que `static`.
- Les éléments enfants sont empilés dans le contexte d'empilement de leur parent le plus proche avec une valeur `position` autre que `static`.

```
/* Exemple 1 */
.parent {
  position: relative;
  z-index: 1;
}
.child {
  position: absolute;
  z-index: 2;
}
```

Dans l'exemple ci-dessus, l'élément `.child` sera empilé au-dessus de l'élément `.parent`, car il a une valeur `position` différente de `static`.

Variables CSS (propriétés personnalisées)

Les variables CSS, également connues sous le nom de propriétés personnalisées, permettent de définir des valeurs spécifiques à réutiliser dans tout votre CSS.

- Définissez des variables avec une syntaxe `--nom-variable: valeur`.
- Utilisez `var(--nom-variable)` pour accéder à la valeur.

```
:root {
  --primary-color: #3498db;
}
body {
  background-color: var(--primary-color);
}
```

Gestion des performances et optimisation

Un CSS propre et bien organisé peut améliorer les performances de rendu de votre site. Quelques astuces pour optimiser votre CSS :

- Minimiser le CSS: Utilisez des outils comme CSSNano pour réduire la taille de votre fichier CSS.
- Utiliser des préprocesseurs: Utilisez des outils comme Sass pour écrire du CSS plus efficacement.

- Evitez les sélecteurs universels (`* {}`) car ils peuvent être coûteux en termes de performance.
- Utilisez le chargement non bloquant: Chargez le CSS non essentiel de manière asynchrone pour éviter de bloquer le rendu de la page.
- Testez les performances: Utilisez des outils comme Lighthouse pour mesurer et optimiser les performances de votre site.

Avec ces astuces et techniques, vous serez mieux équipé pour écrire un CSS efficace et performant.

Ressources et outils

Découvrir et maîtriser les ressources et outils disponibles peut grandement améliorer votre efficacité en tant que développeur CSS. Le développement CSS est facilité par une multitude de ressources, de frameworks et d'outils conçus pour améliorer l'efficacité, la compatibilité et la performance de votre code.

Frameworks CSS

Les frameworks CSS fournissent des structures préconçues et des composants stylistiques pour accélérer le développement.

- **Bootstrap** : Un des frameworks CSS les plus populaires. Il offre un système de grille réactif, des composants prédéfinis et des utilitaires pour un développement rapide. Site officiel
- **Tailwind** : Un framework d'utilitaires en premier, où les classes sont utilisées pour construire des interfaces rapidement sans quitter votre HTML. Site officiel
- **Bulma** : Un framework CSS basé sur des flexbox avec un système de grille moderne et un ensemble de modificateurs. Site officiel

Outils de post-traitement

Les outils de post-traitement transforment votre CSS, généralement pour ajouter de la compatibilité entre navigateurs ou pour introduire des fonctionnalités que les navigateurs ne supportent pas encore.

- **PostCSS** : Un outil pour transformer le CSS avec des plugins JavaScript, comme l'ajout de variables ou l'introduction de nouvelles syntaxes. Site officiel
- **Autoprefixer** : Un plugin PostCSS qui ajoute des préfixes aux règles CSS en utilisant les données de Can I Use pour assurer la compatibilité entre navigateurs. Github Repository

Outils de test et de débogage

- **DevTools** : Les outils de développement intégrés à la plupart des navigateurs modernes (Chrome, Firefox, Safari) offrent des fonctionnalités puissantes pour inspecter, tester et déboguer le CSS.
- **CSS Lint** : Un outil qui analyse votre CSS pour trouver des erreurs, des incohérences ou des violations des bonnes pratiques. Site officiel

Se tenir au courant des dernières ressources et outils est crucial pour rester compétitif et efficace en tant que développeur. Investissez du temps pour explorer, apprendre et intégrer ces outils dans votre flux de travail.