Chapitre 2 : CSS - Approfondissement

1. Définition détaillée du CSS

CSS (Cascading Style Sheets) est le langage de style du web. Il permet de transformer le HTML brut en pages web visuellement attrayantes. Plus qu'un simple outil de mise en forme, CSS permet de :

- Contrôler l'apparence visuelle (couleurs, typographie, espacements)
- Créer des mises en page complexes et responsives
- Ajouter des animations et transitions
- Adapter l'affichage selon le périphérique (mobile, tablette, desktop)

Exemple vivant :
Sans CSS, toutes les pages web ressembleraient à un simple document texte noir sur fond blanc. CSS transforme ces structures HTML en interfaces modernes et engageantes.

---

2. Méthodes d'intégration du CSS - Approfondi

Pourquoi utiliser différentes méthodes d'intégration CSS ?

- **Méthode externe** : privilégiez cette méthode pour des projets de taille moyenne à grande, car elle sépare le contenu HTML du style, facilite la maintenance, et optimise le chargement via la mise en cache.
- **Méthode interne** : utile pour des styles spécifiques à une seule page (par exemple, une page avec un design très différent du reste du site), ou lors de tests rapides durant le développement.
- **Méthode inline** : à réserver à des modifications ponctuelles et isolées ; attention, elle alourdit le HTML et complique la maintenance sur le long terme.

Astuce : Sur les projets professionnels ou collaboratifs, préférez toujours la méthode externe pour garder le code propre et évolutif.

Exemple récapitulatif : - Feuille externe : idéale pour la cohérence globale. - Style interne : dépannage ou pages spéciales. - Inline : ajustement ou cas d'urgence uniquement.

Les trois méthodes d'intégration :

1. CSS Externe (recommandé) :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <!-- Contenu -->
</body>
</html>
```

Avantages : - Réutilisable sur plusieurs pages - Meilleure organisation et maintenance - Mise en cache par le navigateur

2. CSS Interne :

```
<head>
    <style>
        body {
            background-color: #f0f0f0;
            font-family: Arial, sans-serif;
        }
        h1 {
            color: #2c3e50;
        }
    </style>
</head>
```

Utilisation : Pour des styles spécifiques à une seule page.

3. CSS Inline :

```
<p style="color: red; font-size: 16px;">Texte stylisé</p>
```

Note : À éviter sauf cas particuliers (emails HTML, tests rapides).

---

3. Sélecteurs CSS - Guide complet

> **Définition :**
> Un sélecteur CSS sert à cibler un ou plusieurs éléments HTML afin de leur appliquer des styles. Son rôle est essentiel pour déterminer à *quel* contenu les règles CSS vont s'appliquer.

Types principaux de sélecteurs : - **Sélecteur d'élément** : cible tous les éléments d'un certain type (`p` , `h1` , `div` , etc.). - **Sélecteur de classe (`.ma-classe`)** : cible les éléments ayant une classe spécifique (réutilisable). - **Sélecteur d'ID (`#mon-id`)** : cible un élément unique ayant un identifiant spécifique. - **Sélecteur universel (`*`)** : cible tous les éléments de la page. - **Sélecteur d'attribut (`input[type="text"]`)** : cible les éléments selon leurs attributs. - **Sélecteurs combinés** : permettent de cibler des éléments selon leur structure ou relation entre eux (`nav > ul` , `article p` , etc.). - **Pseudo-classes/pseudo-éléments (`:hover` , `::before`)** : ciblent selon des états ou des parties particulières d'un élément.

Le choix du sélecteur influence la portée et la spécificité des styles appliqués.
Astuce : Privilégiez l'usage de classes pour appliquer des styles réutilisables, plutôt que les IDs ou les styles inline.

> **À retenir :**
> Bien choisir son sélecteur rend le code plus lisible, modulaire et performant.

Sélecteurs de base :

```
/* Sélecteur d'élément */
p {
```

```
    color: blue;
}

/* Sélecteur de classe */
.mon-conteneur {
    background-color: #f5f5f5;
}

/* Sélecteur d'ID */
#header {
    height: 80px;
}

/* Sélecteur universel */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

Sélecteurs avancés :

```
/* Sélecteur descendant */
article p {
    line-height: 1.6;
}

/* Sélecteur enfant direct */
nav > ul {
    list-style: none;
}

/* Sélecteur d'attribut */
input[type="text"] {
    border: 1px solid #ccc;
```

```
}

/* Pseudo-classes */
a:hover {
    color: red;
}

button:active {
    transform: scale(0.95);
}

li:first-child {
    font-weight: bold;
}

li:nth-child(odd) {
    background-color: #f9f9f9;
}

/* Pseudo-éléments */
p::first-letter {
    font-size: 2em;
    font-weight: bold;
}

p::before {
    content: "→ ";
    color: blue;
}
```

---

4. Le modèle de boîte (Box Model) - Détaillé

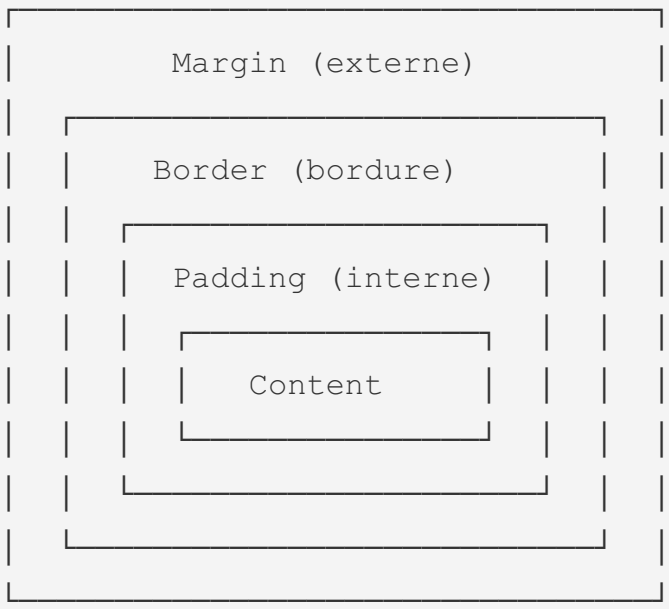
Qu'est-ce que le modèle de boîte CSS ?

Le **modèle de boîte CSS** (Box Model) est un concept fondamental pour la mise en page des éléments HTML. Il définit comment l'espace est attribué autour des éléments et comment ceux-ci interagissent entre eux. Chaque élément en CSS est considéré comme une boîte rectangulaire composée de plusieurs couches :

- **Content (Contenu)** : la zone où le texte et les images apparaissent.
- **Padding (Marge intérieure)** : espace entre le contenu et la bordure.
- **Border (Bordure)** : cadre autour du padding et du contenu.
- **Margin (Marge extérieure)** : espace autour de la bordure, séparant l'élément des autres éléments.

Il est crucial de bien comprendre ce modèle pour aligner, espacer et dimensionner correctement ses éléments dans une page web.

Visualisation :



Remarque : La propriété `box-sizing` permet de choisir la façon dont la largeur et la hauteur sont calculées : - `content-box` (valeur par défaut) : la taille ne comprend que le contenu, le padding et la bordure s'ajoutent. - `border-box` : la taille inclut le padding et la bordure.

Cela influence grandement la manière dont les dimensions et l'agencement des éléments s'effectuent.

Anatomie complète :

```
.box {
    /* Contenu */
    width: 300px;
    height: 200px;

    /* Marge interne */
    padding: 20px;           /* Tous les côtés */
    padding: 10px 20px;      /* Vertical | Horizontal */
    padding: 10px 15px 20px 25px; /* Haut | Droite | Bas | Gauche */

    /* Bordure */
    border: 2px solid #333;
    border-radius: 10px;     /* Coins arrondis */

    /* Marge externe */
    margin: 20px;
    margin: 0 auto;          /* Centrage horizontal */

    /* Box-sizing */
    box-sizing: border-box;  /* Inclut padding et border dans width/heigh
}
```

---

5. Mise en page avec Flexbox - Technique approfondie

Qu'est-ce que Flexbox ? Pourquoi l'utiliser ?

Flexbox (pour *Flexible Box Layout*) est un module CSS moderne conçu pour faciliter l'agencement, l'alignement et la distribution d'espace entre les éléments d'un conteneur, même lorsque leur taille est inconnue ou dynamique.

Pourquoi utiliser Flexbox ?

- Simplifie grandement la création de mises en page complexes (centres verticaux, répartitions dynamiques, colonnes, etc.).
- S'adapte facilement à différents écrans (responsive).
- Gère automatiquement les débordements ou les espaces inutilisés.
- Évite les hacks (float, inline-block, margin negative...) du CSS traditionnel.

Concepts clefs :

- **Conteneur flex** : l'élément parent avec `display: flex`.
- **Éléments flex** : les enfants directs du conteneur.
- Axes principaux :
  - **Axe principal** (`flex-direction`) : direction selon laquelle les éléments sont placés.
  - **Axe transversal** : perpendiculaire à l'axe principal.

Schéma illustratif du modèle Flexbox :

```
| <---- Axe principal (row) ----> |
| [item 1] [item 2] [item 3] ...  |
  ^
  |
Axe transversal (column)
```

> **À retenir :** Flexbox apporte plus de souplesse, de possibilités d'alignement et de réactivité que le modèle de boîte traditionnel.

Exemple de cas d'usage courant : - Centrer parfaitement un bouton, une modal, ou un loader :
Plus besoin de calculs compliqués, on utilise simplement `display: flex; justify-content: center; align-items: center;`.

Conteneur Flex :

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

```
    /* Direction */
    flex-direction: row;        /* row | row-reverse | column | column-re

    /* Alignement horizontal (axe principal) */
    justify-content: center;    /* flex-start | flex-end | center | space

    /* Alignement vertical (axe transversal) */
    align-items: center;        /* flex-start | flex-end | center | stret

    /* Retour à la ligne */
    flex-wrap: wrap;           /* nowrap | wrap | wrap-reverse */

    /* Espacement */
    gap: 20px;                 /* Espace entre éléments */
}
```

Éléments Flex :

```
.item {
    /* Facteur de croissance */
    flex-grow: 1;

    /* Facteur de rétrécissement */
    flex-shrink: 1;

    /* Taille de base */
    flex-basis: 200px;

    /* Raccourci */
    flex: 1 1 200px;  /* grow | shrink | basis */

    /* Alignement individuel */
```

```
    align-self: flex-end;
}
```

Exemple pratique - Navigation responsive :

```
nav {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 20px;
    background-color: #2c3e50;
}

nav ul {
    display: flex;
    gap: 30px;
    list-style: none;
}

nav a {
    color: white;
    text-decoration: none;
    transition: color 0.3s;
}

nav a:hover {
    color: #3498db;
}
```

---

6. Mise en page avec Grid - Guide expert

> **Note de cours :**
> CSS Grid est une méthode puissante pour concevoir des mises en page en

> deux dimensions (lignes et colonnes). Contrairement à Flexbox qui gère surtout l'axe principal, Grid vous permet de positionner précisément des éléments aussi bien horizontalement que verticalement, de créer des zones (areas) réutilisables, et d'adapter simplement la structure de votre site aux différents formats d'écran.
> À retenir : - Définissez les colonnes et lignes du conteneur avec `grid-template-columns` et `grid-template-rows`. - Utilisez les propriétés `grid-column`, `grid-row` ou `grid-area` sur les enfants pour les placer précisément. - Les « grid areas » offrent un contrôle sémantique puissant pour structurer vos pages. - Grid et Flexbox peuvent parfaitement être combinés pour des interfaces complexes et modulaires.

N'hésitez pas à manipuler l'inspecteur de votre navigateur pour visualiser le panneau CSS Grid (souvent disponible dans les DevTools).

Conteneur Grid :

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

    /* Définition des colonnes */
    grid-template-columns: 200px 1fr 200px;  /* 3 colonnes : fixe | flexi
    grid-template-columns: repeat(3, 1fr);   /* 3 colonnes égales */
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr)); /* Respo

    /* Définition des lignes */
    grid-template-rows: 80px auto 60px;

    /* Espacement */
    gap: 20px;                    /* Ou grid-gap */
    column-gap: 20px;
    row-gap: 15px;

    /* Zones nommées */
    grid-template-areas:
        "header header header"
```

```
        "sidebar main main"
        "footer footer footer";
}
```

Éléments Grid :

```
.item1 {
    /* Positionnement par numéro */
    grid-column: 1 / 3;          /* De la colonne 1 à 3 */
    grid-row: 1 / 2;

    /* Ou raccourci */
    grid-area: 1 / 1 / 2 / 3;    /* row-start / col-start / row-end / co
}

.header {
    /* Positionnement par zone nommée */
    grid-area: header;
}
```

Layout complet avec Grid :

```
.page-layout {
    display: grid;
    grid-template-columns: 250px 1fr;
    grid-template-rows: 80px 1fr 60px;
    grid-template-areas:
        "header header"
        "sidebar main"
        "footer footer";
    min-height: 100vh;
    gap: 0;
}
```

```
.header { grid-area: header; background: #2c3e50; }
.sidebar { grid-area: sidebar; background: #ecf0f1; }
.main { grid-area: main; padding: 40px; }
.footer { grid-area: footer; background: #34495e; }
```

---

7. Positionnement CSS - Techniques avancées

> 💡 **Note :**
> Le positionnement avancé en CSS permet d'aller au-delà du flux normal du document afin de créer des layouts complexes. En combinant différents types de positionnement (`relative` , `absolute` , `fixed` , `sticky`), il est possible de superposer des éléments, créer des barres de navigation fixes, ou garder des sections visibles lors du défilement de la page.
> Il est important de bien comprendre le contexte de positionnement (par rapport au parent ou à la fenêtre) pour maîtriser l'empilement et l'agencement des éléments sur une page.

Types de positionnement :

```
/* Static (par défaut) */
.static {
    position: static;
}

/* Relative */
.relative {
    position: relative;
    top: 20px;         /* Décalage par rapport à sa position normale */
    left: 30px;
    z-index: 1;        /* Ordre d'empilement */
}

/* Absolute */
.absolute {
```

```
    position: absolute;
    top: 0;
    right: 0;
    /* Positionné par rapport au premier parent non-static */
}

/* Fixed */
.fixed {
    position: fixed;
    bottom: 20px;
    right: 20px;
    /* Fixe par rapport à la fenêtre du navigateur */
}

/* Sticky */
.sticky {
    position: sticky;
    top: 0;
    /* Reste fixe après défilement jusqu'à un certain point */
}
```

Exemple pratique - Menu sticky :

```
header {
    position: sticky;
    top: 0;
    background: white;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    z-index: 100;
}
```

---

8. Typographie et texte - Guide complet

Dans cette section, nous allons explorer en détail la gestion de la typographie et du texte en CSS. La typographie joue un rôle clé dans le rendu professionnel et lisible d'un site web. Grâce aux propriétés CSS, il est possible de :

- Choisir la famille de police adaptée pour renforcer l'identité visuelle.
- Contrôler la taille, le poids (gras), le style (italique), et la hauteur de ligne pour améliorer la lisibilité.
- Modifier l'espacement entre les lettres ou les mots pour obtenir un rendu aéré ou compact.
- Appliquer des effets graphiques comme les ombres, les soulignements ou la transformation des textes (ex : texte en majuscule).
- Aligner, justifier, ou indenter le texte pour mieux structurer le contenu.

Exemples pratiques et bonnes pratiques : - Privilégiez les polices lisibles pour le texte principal (sans-serif pour les interfaces modernes). - Utilisez les unités relatives (`em` , `rem`) pour que la taille du texte s'adapte aisément selon les appareils et préférences de l'utilisateur. - Veillez à un bon contraste entre le texte et l'arrière-plan. - Testez le rendu sur différents navigateurs et résolutions pour garantir l'accessibilité.

Cette section permettra de maîtriser tous les aspects de la typographie CSS pour obtenir des interfaces esthétiques et confortables à lire.

Propriétés de police :

```
body {
    /* Famille de polices */
    font-family: 'Roboto', Arial, sans-serif;

    /* Taille */
    font-size: 16px;          /* px | em | rem | % */

    /* Poids */
    font-weight: 400;         /* 100-900 | normal | bold */

    /* Style */
```

```
    font-style: italic;       /* normal | italic | oblique */

    /* Hauteur de ligne */
    line-height: 1.6;

    /* Espacement des lettres */
    letter-spacing: 0.5px;

    /* Espacement des mots */
    word-spacing: 2px;
}
```

Formatage du texte :

```
p {
    /* Alignement */
    text-align: center;       /* left | center | right | justify */

    /* Transformation */
    text-transform: uppercase; /* uppercase | lowercase | capitalize */

    /* Décoration */
    text-decoration: underline; /* none | underline | line-through */

    /* Ombre */
    text-shadow: 2px 2px 4px rgba(0,0,0,0.3);

    /* Indentation */
    text-indent: 30px;

    /* Césure */
    word-wrap: break-word;
    overflow-wrap: break-word;
}
```

Import de Google Fonts :

```
<head>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300
</head>
```

---

9. Couleurs et arrière-plans - Techniques avancées

> **Note de cours :**
>
> Les couleurs et arrière-plans en CSS permettent de créer des designs visuellement attractifs et hiérarchisés. En utilisant différentes méthodes pour définir les couleurs (noms, codes hexadécimaux, RGB, HSL) et en combinant des arrière-plans complexes (images, dégradés, multiples couches), il est possible d'obtenir des effets avancés adaptés à tout type de projet web. La maîtrise de ces propriétés est essentielle pour concevoir des interfaces modernes, lisibles et esthétiques.

Formats de couleurs :

```
.element {
    /* Nom de couleur */
    color: red;

    /* Hexadécimal */
    color: #ff0000;
    color: #f00;              /* Forme courte */

    /* RGB */
    color: rgb(255, 0, 0);

    /* RGBA (avec transparence) */
    color: rgba(255, 0, 0, 0.5);
```

```
    /* HSL */
    color: hsl(0, 100%, 50%);

    /* HSLA */
    color: hsla(0, 100%, 50%, 0.5);
}
```

Arrière-plans complexes :

```
.background {
    /* Couleur de fond */
    background-color: #f0f0f0;

    /* Image de fond */
    background-image: url('image.jpg');
    background-repeat: no-repeat;     /* repeat | repeat-x | repeat-y | r
    background-position: center;      /* top | bottom | left | right | ce
    background-size: cover;           /* auto | cover | contain | px | %
    background-attachment: fixed;     /* scroll | fixed */

    /* Raccourci */
    background: url('bg.jpg') no-repeat center/cover fixed;

    /* Dégradés linéaires */
    background: linear-gradient(to right, #667eea, #764ba2);
    background: linear-gradient(45deg, red, yellow, green);

    /* Dégradés radiaux */
    background: radial-gradient(circle, #667eea, #764ba2);

    /* Multiples arrière-plans */
    background:
        linear-gradient(rgba(0,0,0,0.5), rgba(0,0,0,0.5)),
```

```
        url('image.jpg') center/cover;
}
```

---

10. Responsive Design - Guide complet

Qu'est-ce que le Responsive Design ?

Le **Responsive Design** (ou design réactif) est une approche de conception web qui vise à rendre les pages Web consultables de façon optimale quel que soit le terminal utilisé : ordinateur de bureau, tablette, smartphone, etc. L'objectif est d'offrir une expérience utilisateur fluide et adaptée à toutes les résolutions d'écran, sans avoir à développer plusieurs versions d'un même site.

Principes clés du responsive design :

- **Utilisation de grilles fluides:** Les éléments s'adaptent proportionnellement à la largeur de l'écran, grâce aux unités relatives (%, rem, vw, etc.).
- **Médias flexibles :** Les images, vidéos et contenus multimédia sont redimensionnés (ex: `max-width: 100%`) pour ne jamais dépasser la largeur du conteneur.
- **Media queries:** Permettent d'appliquer des styles CSS spécifiques selon la taille de l'écran, l'orientation ou la résolution.

Avantages :

- Navigation confortable sur tous supports (mobile, tablette, desktop).
- Meilleur référencement (SEO), Google privilégiant le mobile-first.
- Maintenance facilitée : un seul code source à gérer.

Bonnes pratiques :

- Penser "Mobile First" : concevoir d'abord pour les petits écrans, puis enrichir pour les grands écrans.
- Tester régulièrement sur différents appareils et résolutions.
- Privilégier des textes lisibles, des boutons facilement cliquables et une navigation intuitive.

> **Astuce:** Utilisez les outils de développement des navigateurs (ex. Responsive Design Mode dans Chrome/Firefox) pour simuler différents écrans durant la conception de vos sites !

Media Queries essentielles :

```
/* Mobile First (recommandé) */
/* Styles de base pour mobile */
body {
    font-size: 14px;
}

/* Tablette */
@media (min-width: 768px) {
    body {
        font-size: 16px;
    }

    .container {
        max-width: 720px;
    }
}

/* Desktop */
@media (min-width: 1024px) {
    body {
        font-size: 18px;
    }

    .container {
        max-width: 960px;
    }
}

/* Large Desktop */
```

```
@media (min-width: 1440px) {
    .container {
        max-width: 1200px;
    }
}

/* Desktop First (alternative) */
@media (max-width: 768px) {
    .menu {
        flex-direction: column;
    }
}

/* Orientation */
@media (orientation: landscape) {
    .hero {
        height: 100vh;
    }
}

/* Préférence utilisateur */
@media (prefers-color-scheme: dark) {
    body {
        background: #1a1a1a;
        color: white;
    }
}
```

Unités responsives :

```
.responsive {
    /* Unités relatives */
    width: 80%;               /* Pourcentage du parent */
    font-size: 1.5rem;        /* Relatif à la taille racine */
    padding: 2em;             /* Relatif à la taille de police de l'éléme
```

```
    /* Viewport units */
    width: 100vw;             /* 100% de la largeur du viewport */
    height: 100vh;            /* 100% de la hauteur du viewport */
    font-size: 5vw;           /* 5% de la largeur du viewport */

    /* Fonctions min/max/clamp */
    width: min(90%, 1200px);  /* Le plus petit entre 90% et 1200px */
    width: max(300px, 50%);   /* Le plus grand entre 300px et 50% */
    font-size: clamp(1rem, 2.5vw, 2rem); /* Min | Idéal | Max */
}
```

---

11. Animations et Transitions - Techniques avancées

Notions avancées sur les animations CSS

- **Différences entre transition et animation** :
- Les **transitions** nécessitent une action de l'utilisateur (hover, focus, etc.) pour déclencher le changement d'état ; elles ne se répètent pas.

- Les **animations** via `@keyframes` peuvent s'exécuter automatiquement à l'arrivée de l'élément sur la page, et permettent des étapes multiples, des boucles, etc.

- **Propriétés utiles pour les animations :**

- `animation-name` : le nom du `@keyframes` à utiliser.
- `animation-duration` : durée totale de l'animation (ex : `2s`).
- `animation-delay` : délai avant le démarrage (ex : `0.5s`).
- `animation-iteration-count` : nombre de répétitions ou `infinite`.
- `animation-direction` : sens de lecture (`normal`, `reverse`, `alternate`).

- `animation-fill-mode` : comportement avant/après (`forwards`, `backwards`, `both`).

- **Chaining & composition :**
  Vous pouvez combiner plusieurs animations sur un même élément ou enchaîner transitions et animations.

- **Performance :**

- Privilégier l'animation de `transform` et `opacity` pour de meilleures performances.
- Éviter d'animer des propriétés qui forcent le reflow (`width` , `height` , `top` , `left`).

Exemple de syntaxe complète :

```
.animated-box {
    animation: bounce 1s 0.2s 3 alternate both;
}

@keyframes bounce {
    0% { transform: translateY(0);}
    30% { transform: translateY(-50px);}
    100% { transform: translateY(0);}
}
```

- Ici, l'élément `.animated-box` fait l'animation `bounce` pendant `1s` , attend `0.2s` avant de démarrer, répète 3 fois en alternant le sens, et garde le style final après la fin.

- **Tips :**
  Pensez à utiliser les outils de dev des navigateurs pour déboguer et ajuster vos animations.

Transitions :

```
button {
    background-color: #3498db;
    color: white;
    padding: 10px 20px;
```

```
    border: none;
    cursor: pointer;

    /* Transition sur toutes les propriétés */
    transition: all 0.3s ease;

    /* Ou transitions spécifiques */
    transition:
        background-color 0.3s ease,
        transform 0.2s ease-in-out;
}

button:hover {
    background-color: #2980b9;
    transform: translateY(-2px);
    box-shadow: 0 5px 15px rgba(0,0,0,0.2);
}

/* Fonctions de temporisation */
.element {
    transition-timing-function: ease;        /* ease | linear | ease-in
    transition-timing-function: cubic-bezier(0.68, -0.55, 0.265, 1.55); /
}
```

Animations avec @keyframes :

```
/* Définition de l'animation */
@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
```

```
    }
}

/* Avec étapes multiples */
@keyframes pulse {
    0% {
        transform: scale(1);
    }
    50% {
        transform: scale(1.05);
    }
    100% {
        transform: scale(1);
    }
}

/* Application de l'animation */
.fade-in {
    animation-name: fadeIn;
    animation-duration: 1s;
    animation-timing-function: ease-out;
    animation-delay: 0.5s;
    animation-iteration-count: 1;      /* infinite | nombre */
    animation-direction: normal;       /* normal | reverse | alternate */
    animation-fill-mode: forwards;     /* none | forwards | backwards |

    /* Raccourci */
    animation: fadeIn 1s ease-out 0.5s forwards;
}

.pulse {
    animation: pulse 2s ease-in-out infinite;
}
```

Animations complexes :

```
@keyframes rotate-and-scale {
    0% {
        transform: rotate(0deg) scale(1);
        background-color: #3498db;
    }
    50% {
        transform: rotate(180deg) scale(1.2);
        background-color: #e74c3c;
    }
    100% {
        transform: rotate(360deg) scale(1);
        background-color: #3498db;
    }
}

.animated-box {
    width: 100px;
    height: 100px;
    animation: rotate-and-scale 3s ease-in-out infinite;
}
```

---

12. Variables CSS (Custom Properties) - Guide complet

> **Note de cours :**
> Les variables CSS, appelées aussi *Custom Properties*, permettent de définir des valeurs réutilisables dans toute votre feuille de style.
> Elles facilitent la maintenance, rendent le code plus lisible, et permettent de créer rapidement des thèmes ou de modifier facilement certains aspects visuels d'un site en ne changeant la valeur qu'à un seul endroit.

> **À retenir :** - Les variables CSS sont déclarées avec `--nom-variable` dans un sélecteur, souvent `:root` pour les rendre globales. - Elles s'utilisent avec la fonction `var(--nom-variable)` partout où une valeur CSS est attendue. - Il est possible de leur donner une valeur par défaut : `var(--nom-variable, valeur-de-repli)`.

Déclaration et utilisation :

```
:root {
    /* Déclaration globale */
    --color-primary: #3498db;
    --color-secondary: #2ecc71;
    --color-accent: #e74c3c;

    --font-main: 'Roboto', sans-serif;
    --font-heading: 'Montserrat', sans-serif;

    --spacing-small: 8px;
    --spacing-medium: 16px;
    --spacing-large: 32px;

    --border-radius: 4px;
    --transition-speed: 0.3s;
}

/* Utilisation */
button {
    background-color: var(--color-primary);
    font-family: var(--font-main);
    padding: var(--spacing-medium);
    border-radius: var(--border-radius);
    transition: all var(--transition-speed);
}

/* Valeur par défaut */
.element {
```

```
    color: var(--color-text, black); /* Utilise black si --color-text n'e
}
```

Variables contextuelles :

```
.card {
    --card-bg: white;
    --card-shadow: 0 2px 4px rgba(0,0,0,0.1);

    background: var(--card-bg);
    box-shadow: var(--card-shadow);
}

.card.dark {
    --card-bg: #2c3e50;
    --card-shadow: 0 2px 4px rgba(255,255,255,0.1);
}
```

Thème complet avec variables :

```
:root {
    --bg-primary: white;
    --bg-secondary: #f5f5f5;
    --text-primary: #333;
    --text-secondary: #666;
}

[data-theme="dark"] {
    --bg-primary: #1a1a1a;
    --bg-secondary: #2d2d2d;
    --text-primary: #f5f5f5;
    --text-secondary: #aaa;
}
```

```
body {
    background-color: var(--bg-primary);
    color: var(--text-primary);
}
```

Exercices approfondis avec corrections

Exercice 1 : Carte de profil complète

Consigne : Créez une carte de profil avec : 1. Photo de profil circulaire 2. Nom et titre avec typographie personnalisée 3. Bouton de contact avec animation au survol 4. Mise en page centrée avec Flexbox

Correction :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Carte de Profil</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%)
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif,
```

```
        }

        .profile-card {
            background: white;
            border-radius: 20px;
            padding: 40px;
            text-align: center;
            box-shadow: 0 10px 30px rgba(0,0,0,0.3);
            max-width: 350px;
        }

        .profile-image {
            width: 150px;
            height: 150px;
            border-radius: 50%;
            object-fit: cover;
            border: 5px solid #667eea;
            margin-bottom: 20px;
        }

        .profile-name {
            font-size: 28px;
            font-weight: bold;
            color: #2c3e50;
            margin-bottom: 10px;
        }

        .profile-title {
            font-size: 16px;
            color: #7f8c8d;
            margin-bottom: 30px;
        }

        .contact-btn {
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%)
            color: white;
```

```
            border: none;
            padding: 12px 30px;
            border-radius: 25px;
            font-size: 16px;
            cursor: pointer;
            transition: all 0.3s ease;
        }

        .contact-btn:hover {
            transform: translateY(-3px);
            box-shadow: 0 7px 20px rgba(102, 126, 234, 0.4);
        }

        .contact-btn:active {
            transform: translateY(-1px);
        }
    </style>
</head>
<body>
    <div class="profile-card">
        <img src="https://i.pravatar.cc/150?img=12" alt="Photo de profil"
        <h2 class="profile-name">Marie Dubois</h2>
        <p class="profile-title">Développeuse Web Full-Stack</p>
        <button class="contact-btn">Me Contacter</button>
    </div>
</body>
</html>
```

Exercice 2 : Layout responsive avec Grid

Consigne : Créez une page d'articles de blog avec : 1. Header fixe 2. Sidebar avec catégories 3. Grille d'articles responsive (3 colonnes desktop, 2 tablette, 1 mobile) 4. Footer

Correction :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    <title>Blog Layout</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: Arial, sans-serif;
            line-height: 1.6;
        }

        .page-layout {
            display: grid;
            grid-template-columns: 250px 1fr;
            grid-template-rows: 80px 1fr 60px;
            grid-template-areas:
                "header header"
                "sidebar main"
                "footer footer";
            min-height: 100vh;
        }

        .header {
            grid-area: header;
            background: #2c3e50;
            color: white;
            display: flex;
            align-items: center;
```

```
            padding: 0 30px;
            position: sticky;
            top: 0;
            z-index: 100;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        }

        .sidebar {
            grid-area: sidebar;
            background: #ecf0f1;
            padding: 20px;
        }

        .sidebar h3 {
            margin-bottom: 15px;
            color: #2c3e50;
        }

        .sidebar ul {
            list-style: none;
        }

        .sidebar li {
            padding: 10px;
            margin-bottom: 5px;
            background: white;
            border-radius: 5px;
            cursor: pointer;
            transition: all 0.2s;
        }

        .sidebar li:hover {
            background: #3498db;
            color: white;
            transform: translateX(5px);
        }
```

```
.main {
    grid-area: main;
    padding: 30px;
}

.articles-grid {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 30px;
}

.article-card {
    background: white;
    border-radius: 10px;
    overflow: hidden;
    box-shadow: 0 2px 10px rgba(0,0,0,0.1);
    transition: transform 0.3s, box-shadow 0.3s;
}

.article-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 5px 20px rgba(0,0,0,0.2);
}

.article-image {
    width: 100%;
    height: 200px;
    object-fit: cover;
}

.article-content {
    padding: 20px;
}

.article-title {
```

```
            font-size: 20px;
            margin-bottom: 10px;
            color: #2c3e50;
        }

        .article-excerpt {
            color: #7f8c8d;
            font-size: 14px;
        }

        .footer {
            grid-area: footer;
            background: #34495e;
            color: white;
            display: flex;
            align-items: center;
            justify-content: center;
        }

        /* Responsive */
        @media (max-width: 1024px) {
            .articles-grid {
                grid-template-columns: repeat(2, 1fr);
            }
        }

        @media (max-width: 768px) {
            .page-layout {
                grid-template-columns: 1fr;
                grid-template-areas:
                    "header"
                    "main"
                    "sidebar"
                    "footer";
            }
```

```
            .articles-grid {
                grid-template-columns: 1fr;
            }

            .sidebar {
                border-top: 2px solid #bdc3c7;
            }
        }
    </style>
</head>
<body>
    <div class="page-layout">
        <header class="header">
            <h1>Mon Blog</h1>
        </header>

        <aside class="sidebar">
            <h3>Catégories</h3>
            <ul>
                <li>Technologie</li>
                <li>Design</li>
                <li>Développement</li>
                <li>Tutoriels</li>
                <li>Actualités</li>
            </ul>
        </aside>

        <main class="main">
            <div class="articles-grid">
                <article class="article-card">
                    <img src="https://picsum.photos/400/200?random=1" alt
                    <div class="article-content">
                        <h3 class="article-title">Introduction au CSS Gri
                        <p class="article-excerpt">Découvrez comment crée
                    </div>
                </article>
```

```
                <article class="article-card">
                    <img src="https://picsum.photos/400/200?random=2" alt
                    <div class="article-content">
                        <h3 class="article-title">Flexbox vs Grid</h3>
                        <p class="article-excerpt">Quand utiliser Flexbox
                    </div>
                </article>
                <article class="article-card">
                    <img src="https://picsum.photos/400/200?random=3" alt
                    <div class="article-content">
                        <h3 class="article-title">Animations CSS</h3>
                        <p class="article-excerpt">Créez des animations
                    </div>
                </article>
                <article class="article-card">
                    <img src="https://picsum.photos/400/200?random=4" alt
                    <div class="article-content">
                        <h3 class="article-title">Design Responsive</h3>
                        <p class="article-excerpt">Les meilleures pratiqu
                    </div>
                </article>
                <article class="article-card">
                    <img src="https://picsum.photos/400/200?random=5" alt
                    <div class="article-content">
                        <h3 class="article-title">Variables CSS</h3>
                        <p class="article-excerpt">Simplifiez votre code
                    </div>
                </article>
                <article class="article-card">
                    <img src="https://picsum.photos/400/200?random=6" alt
                    <div class="article-content">
                        <h3 class="article-title">Performance CSS</h3>
                        <p class="article-excerpt">Optimisez vos styles p
                    </div>
                </article>
            </div>
```

```
        </main>

        <footer class="footer">
            <p>&copy; 2024 Mon Blog - Tous droits réservés</p>
        </footer>
    </div>
</body>
</html>
```

Exercice 3 : Animation de chargement

Consigne : Créez un loader animé avec : 1. Animation de rotation continue 2. Utilisation de keyframes 3. Dégradé de couleurs 4. Centrage parfait sur la page

Correction :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Loader Animé</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            display: flex;
            justify-content: center;
            align-items: center;
            min-height: 100vh;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%)
        }
```

```
        .loader-container {
            text-align: center;
        }

        .loader {
            width: 80px;
            height: 80px;
            border: 8px solid rgba(255, 255, 255, 0.3);
            border-top: 8px solid white;
            border-radius: 50%;
            animation: spin 1s linear infinite;
        }

        @keyframes spin {
            0% {
                transform: rotate(0deg);
            }
            100% {
                transform: rotate(360deg);
            }
        }

        .loader-text {
            color: white;
            font-family: Arial, sans-serif;
            font-size: 18px;
            margin-top: 20px;
            animation: pulse 1.5s ease-in-out infinite;
        }

        @keyframes pulse {
            0%, 100% {
                opacity: 1;
            }
            50% {
                opacity: 0.5;
```

```
        }
    }

    /* Loader alternatif avec points */
    .dots-loader {
        display: flex;
        gap: 10px;
        margin-top: 30px;
    }

    .dot {
        width: 15px;
        height: 15px;
        background: white;
        border-radius: 50%;
        animation: bounce 1.4s infinite ease-in-out;
    }

    .dot:nth-child(1) {
        animation-delay: 0s;
    }

    .dot:nth-child(2) {
        animation-delay: 0.2s;
    }

    .dot:nth-child(3) {
        animation-delay: 0.4s;
    }

    @keyframes bounce {
        0%, 80%, 100% {
            transform: scale(0);
            opacity: 0.5;
        }
        40% {
```

```
                transform: scale(1);
                opacity: 1;
            }
        }
    </style>
</head>
<body>
    <div class="loader-container">
        <div class="loader"></div>
        <p class="loader-text">Chargement en cours...</p>
        <div class="dots-loader">
            <div class="dot"></div>
            <div class="dot"></div>
            <div class="dot"></div>
        </div>
    </div>
</body>
</html>
```

Bonnes pratiques CSS professionnelles

1. Organisation du code :

```
/* Structure recommandée d'un fichier CSS */

/* ===============================================
   1. Reset et styles de base
   =============================================== */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

```
/* ================================================
   2. Variables CSS
   ================================================ */
:root {
    --color-primary: #3498db;
    --spacing-unit: 8px;
}

/* ================================================
   3. Typographie
   ================================================ */
body {
    font-family: Arial, sans-serif;
    line-height: 1.6;
}

/* ================================================
   4. Layout
   ================================================ */
.container {
    max-width: 1200px;
    margin: 0 auto;
}

/* ================================================
   5. Composants
   ================================================ */
.button { ... }
.card { ... }

/* ================================================
   6. Utilitaires
   ================================================ */
.text-center { text-align: center; }
.mt-1 { margin-top: 8px; }
```

```
/* ==========================================
   7. Media Queries
   ========================================== */
@media (max-width: 768px) { ... }
```

2. Nomenclature BEM :

```
/* Block Element Modifier */

/* Block */
.card { }

/* Element */
.card__title { }
.card__content { }
.card__button { }

/* Modifier */
.card--featured { }
.card__button--primary { }
.card__button--disabled { }
```

3. Performance :

```
/* ✅ BON - Sélecteurs spécifiques */
.navigation-item { }

/* ❌ MAUVAIS - Sélecteurs trop génériques */
* { }
div div div p { }

/* ✅ BON - Éviter les propriétés coûteuses */
.element {
```

```
    transform: translateX(10px); /* GPU accéléré */
}

/* ❌ MAUVAIS */
.element {
    left: 10px; /* Provoque un reflow */
}
```

4. Accessibilité :

```
/* Focus visible pour la navigation au clavier */
button:focus {
    outline: 2px solid #3498db;
    outline-offset: 2px;
}

/* Masquer visuellement mais garder pour lecteurs d'écran */
.sr-only {
    position: absolute;
    width: 1px;
    height: 1px;
    padding: 0;
    margin: -1px;
    overflow: hidden;
    clip: rect(0, 0, 0, 0);
    white-space: nowrap;
    border: 0;
}

/* Contraste suffisant */
.text {
    color: #333;  /* Ratio de contraste >= 4.5:1 */
    background: white;
}
```

5. Maintenabilité :

```
/* ✅ BON - Code commenté */
/*
 * Card Component
 * Description: Carte réutilisable pour afficher du contenu
 * Usage: <div class="card">...</div>
 */
.card {
    background: white;
    border-radius: 8px;
    padding: 20px;
}

/* ✅ BON - Valeurs cohérentes */
:root {
    --spacing-xs: 4px;
    --spacing-sm: 8px;
    --spacing-md: 16px;
    --spacing-lg: 32px;
}

/* ✅ BON - Éviter les magic numbers */
.element {
    padding: var(--spacing-md); /* Au lieu de padding: 16px */
}
```

6. Responsive :

```
/* Mobile First approche */
.element {
    font-size: 16px;
}
```

```
@media (min-width: 768px) {
    .element {
        font-size: 18px;
    }
}

@media (min-width: 1024px) {
    .element {
        font-size: 20px;
    }
}

/* Breakpoints cohérents */
:root {
    --breakpoint-sm: 576px;
    --breakpoint-md: 768px;
    --breakpoint-lg: 992px;
    --breakpoint-xl: 1200px;
}
```

Ressources pour aller plus loin

1. Documentation officielle :

- MDN Web Docs - CSS
- CSS Specification W3C
- Can I Use - Compatibilité navigateurs

2. Outils pratiques :

- **Validateurs :**
- W3C CSS Validator
- **Générateurs :**
- CSS Grid Generator

- Flexbox Generator
- Gradient Generator
- **Éditeurs en ligne :**
- CodePen
- JSFiddle
- CSS Playground

3. Concepts avancés à explorer :

- **CSS Modules** - Isolation des styles
- **Sass/SCSS** - Préprocesseur CSS
- **PostCSS** - Transformation CSS avec plugins
- **CSS-in-JS** - Styles dans JavaScript (Styled Components)
- **Tailwind CSS** - Framework utility-first
- **CSS Houdini** - API bas niveau pour le rendu

4. Sites d'inspiration :

- Awwwards
- CSS Design Awards
- Dribbble
- Behance

5. Pratique et défis :

- CSS Battle - Défis de reproduction visuelle
- 100 Days CSS Challenge
- Frontend Mentor
- Daily UI

Conclusion

Le CSS est un outil puissant qui nécessite de la pratique régulière. Les clés du succès sont :

1. **Pratiquer quotidiennement** - Reproduisez des designs existants
2. **Expérimenter** - Testez de nouvelles propriétés et techniques

3. **Lire le code des autres** - Inspectez les sites que vous aimez
4. **Rester à jour** - Le CSS évolue constamment
5. **Construire des projets réels** - Mettez en pratique vos connaissances

Exercice final recommandé : Créez un portfolio personnel complet en utilisant tous les concepts appris dans ce chapitre. C'est le meilleur moyen de consolider vos connaissances !

Bonne continuation dans votre apprentissage du CSS ! 🎨