Usages des techniques numériques

Maitriser les technologies du web

Module 3 – CSS : Présentation et mise en forme d'une page web

Introduction aux CSS

- Le CSS (**Cascading Style Sheets**) définit l'apparence visuelle des pages web et complète la structure fournie par le HTML.

- En assurant la séparation entre structure (HTML) et style (CSS), on favorise la lisibilité du code, la maintenabilité des interfaces et la cohérence graphique sur l'ensemble d'un site.

- Le CSS joue un rôle central dans l'adaptation d'une interface aux différents supports (ordinateurs, tablettes, smartphones) grâce aux techniques de **responsive design**.

- L'HTML définit le *quoi* : les éléments et leur sens
  tandis que le CSS définit le *comment* : la manière dont ces éléments apparaissent à l'écran.

Sommaire

1. **Principes fondamentaux du CSS** (cascade, héritage, spécificité, unités).....p.04
2. **Où placer le CSS** (externe / interne / inline).....p.10
3. **Reset / Normalize CSS**.....p.11
4. **Sélecteurs et spécificités** (types, classes, identifiants, attributs, structurels, pseudo-classes, pseudo-éléments).....p.14
5. **Parent / enfant / ancêtre** (structure du DOM).....p.27
6. **Box model**.....p.28
7. **Typographie & couleur**.....p.33
8. **Variables avancées / Design Tokens**.....p.41
9. **Positionnement CSS**.....p.44
10. **Transitions & animations**.....p.48
11. **Effets visuels** (ombres, arrondis, dégradés, filtres, arrière-plans) p.51
12. **Flexbox & Grid**.....p.55
13. **Responsive design**.....p.62

1. Principes fondamentaux du CSS

1.1. Rôle du CSS

CSS (Cascading Style Sheets) permet de contrôler la présentation d'une page web :

- couleurs
- polices
- espacements
- alignements
- mise en page (layout)
- interactions visuelles (hover, transitions…)
- responsive design (adaptation mobile)

Rappel :

> HTML structure

> CSS présente

> JavaScript agit

1. Principes fondamentaux du CSS

1.2. Comment fonctionne le CSS ?

Le fonctionnement du CSS repose sur trois principes fondamentaux : **la cascade**, **l'héritage** et **la spécificité**. Ensemble, ils déterminent quels styles sont appliqués lorsqu'un navigateur affiche une page.

1.2.1. La cascade

- La cascade correspond à **l'ordre dans lequel le navigateur lit et applique les règles CSS**.
- Lorsque plusieurs déclarations ciblent un même élément, le navigateur suit ce principe :
  > les feuilles de style du navigateur (styles par défaut) sont appliquées en premier,
  > puis les feuilles externes, internes, et enfin les styles inline,
  > et si deux règles ont exactement la même priorité, **la dernière définie dans le code** l'emporte.
- Cela permet d'écrire des styles généraux, puis de les affiner ou de les corriger plus bas dans la feuille.

1. Principes fondamentaux du CSS

1.2. Comment fonctionne le CSS ?

1.2.2. L'héritage

- Certaines propriétés se transmettent automatiquement aux éléments enfants, comme :
  - la couleur du texte (color),
  - la police (font-family),
  - l'interlignage (line-height).

- D'autres propriétés **ne s'héritent pas**, comme margin, border ou width.

- L'héritage permet de définir des propriétés globales sur <html> ou <body> puis de laisser les éléments internes en bénéficier.

1. Principes fondamentaux du CSS

1.2. Comment fonctionne le CSS ?

1.2.3. La spécificité

- La spécificité détermine la force d'un sélecteur : plus il est précis, plus il a de poids.
- Ordre croissant :
  1. sélecteurs de type (section, article, p, div),
  2. classes (.menu, .actif) et pseudo-classes (:hover, :focus),
  3. identifiants (#hero),
  4. styles inline,
  5. !important (à utiliser avec prudence).
- Si deux règles ciblent le même élément, **celle qui possède la spécificité la plus élevée** s'applique.

1. Principes fondamentaux du CSS

1.3. Les unités CSS

- Le CSS utilise différentes unités pour définir les tailles, espacements, polices ou mises en page.
- Ces unités influencent directement la flexibilité, l'accessibilité et l'adaptation du design aux différents écrans.
- **Pourquoi les unités sont essentielles**
  - Elles influencent le comportement du layout, pour le responsive.
  - Elles permettent d'obtenir des composants adaptatifs et accessibles.
  - Elles évitent les mises en page "rigides" basées uniquement sur les pixels.

1.3.1. Les unités absolues

- **px (pixel)** : unité fixe, indépendante du parent.

  > **Usage :** bordures, ombres, détails fins.

```
article { border: 1px solid #333; }
```

1. Principes fondamentaux du CSS

1.3. Les unités CSS

1.3.2. Les unités relatives (recommandées)

Elles s'adaptent au contexte et favorisent des interfaces modulables.

- **%** : proportionnelle à la taille du conteneur parent
  > utile pour largeurs fluides et images responsives.

```
article { width: 50%; }
```

- **em** : relative à la taille de la police du parent
  > utile pour les composants modulaires

```
body { font-size: 20px; }
p {
  font-size: 1.5em;  /* = 1.5 × 20px = 30px */
}
```

- **rem** : relative à la taille de la racine (html)
  > 1em = 16px, favorise l'accessibilité et une typographie cohérente.

```
p {
  font-size: 1.5rem;  /* = 1.5 × 16px = 24px */
}
```

- **vw / vh** : pourcentage de la fenêtre
  > pratique pour les sections pleine largeur/hauteur.
  - 1vw = 1 % de la largeur de l'écran
  - 1vh = 1% de la hauteur de l'écran.

```
.hero {
  width: 100vw;     /* pleine largeur */
  height: 50vh;     /* 50 % hauteur écran */
}
```

2. Où placer le CSS ?

- **Trois méthodes :**

Méthode	Exemple	Usage
CSS externe	Ficher lié dans <head> en html : <link rel="stylesheet" href="style.css">	Meilleure pratique pour tout site
CSS interne	Style inséré dans <head> en html : <style>p { color: #333; }</style>	Pour tests rapides ou pages isolées
Inline	Style inséré dans la balise html : <p style="color: #333;">	À éviter (pas maintenable)

> Recommandation :

Toujours utiliser **des fichiers CSS externes** pour séparer clairement structure et apparence, comme vu dans le module HTML.

3. Reset / Normalize CSS

3.1. Pourquoi un reset CSS ?

- Chaque navigateur applique par défaut ses propres styles :
  marges sur <body>, tailles de titres, espacements des listes, bordures des formulaires…
  Ces différences créent des variations visuelles d'un navigateur à l'autre.

- **Un reset permet :**
  - d'annuler les styles par défaut,
  - d'obtenir une base cohérente,
  - de démarrer la mise en forme sur des fondations uniformes.

3.2. Reset minimal

- Un reset léger, qui suffit pour la majorité des projets :
  - supprime toutes les marges/paddings par défaut
  - évite les calculs de largeur compliqués
  - rend la mise en page plus prévisible

```
*, *::before, *::after {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}
```

3. Reset / Normalize CSS

3.3. Normalize CSS

- Alternative au reset classique.

> **Normalize ne supprime pas les styles**, mais les rend **cohérents entre navigateurs** :
  - respecte les tailles et marges utiles
  - corrige les incohérences (titres, formulaires, tableaux)
  - préserve l'accessibilité

- On l'utilise via un fichier externe :

```
<link rel="stylesheet" href="normalize.css">
```

- **Avantage :** réduction des divergences sans perdre les styles utiles
  Inconvénient : beaucoup plus long qu'un reset minimal.

3. Reset / Normalize CSS

3.4. Quel choix faire ?

- **Reset minimal**
  > pour un design personnalisé, épuré, et des composants sur mesure.
  > idéal pour un projet éducatif ou professionnel moderne.

- **Normalize**
  > pour un site éditorial ou un projet qui veut garder certains comportements natifs.

- **Bonnes pratiques**
  - Toujours appliquer le reset **tout en haut** de la feuille CSS.
  - Garder un reset simple pour faciliter la lecture du code.
  - Utiliser box-sizing: border-box en global (standard).
  - Éviter les resets trop agressifs sur les formulaires si on veut conserver l'accessibilité native.

4. Sélecteurs et spécificités

- Les sélecteurs permettent d'indiquer quels éléments HTML doivent recevoir un style.
- Chaque sélecteur possède un **poids**, appelé spécificité, qui permet au navigateur de déterminer laquelle de plusieurs règles concurrentes doit s'appliquer.
- Les sélecteurs les plus utilisés se regroupent en **sept grandes familles :**

1. **Sélecteurs de type**
2. **Sélecteurs de classe**
3. **Sélecteurs d'identifiant**
4. **Sélecteurs structurels**
5. **Sélecteurs d'attribut**
6. **Sélecteurs d'état** (pseudo-classes)
7. **Pseudo-éléments**

4. Sélecteurs et spécificités

4.1. Sélecteurs de type *(nom-de-la-balise-html)*

p { … }
h1 { … }
div { … }
header { … }

> **Rôle :** Ciblent tous les éléments d'un même type de balise.

> **Usage :** Définir les styles basiques et globaux : tous les paragraphes, tous les titres, tous les liens, etc.

> **Spécificité : Faible**
C'est l'un des niveaux les plus bas : un sélecteur de type perdra face à une classe ou un identifiant.

4. Sélecteurs et spécificités

4.2. Sélecteurs de classe *(.nom-perso)*

.intro { ... }
.menu-item { ... }
.highlight { ... }
.orange { ... }

> Rôle : Ciblent les éléments portant une classe donnée.
Une classe peut être attribuée à plusieurs éléments.
Plusieurs classes peuvent être attribuées à un élément.

> Usage : Construire des composants, des blocs réutilisables, des mises en forme cohérentes.
C'est la base d'un CSS propre et modulable.

> Spécificité : Moyenne
Supérieure à un type, inférieure à un identifiant. Très bon compromis entre précision et maintenabilité.

4. Sélecteurs et spécificités

4.3. Sélecteurs d'identifiant *(#nom-perso)*

#hero { … }
#features { … }

> Rôle : Ciblent un élément **unique** dans la page.
Un identifiant n'est jamais censé être réutilisé.

> Usage : Plutôt réservé aux sections et pour des besoins techniques (ancres, ARIA, JavaScript).

> Spécificité : Élevée
Un identifiant l'emporte sur une classe ou un type.
À utiliser avec parcimonie pour éviter des conflits difficiles à corriger.

4. Sélecteurs et spécificités

4.4. **Sélecteurs structurels** (basés sur les relations dans le DOM)

- **Descendant :** Style appliqué à un élément situé quelque part à l'intérieur d'un autre.
  header nav a { ... }
- **Enfant direct :** Cible uniquement les éléments qui sont directement contenus dans un autre élément.
  nav > a { ... }
- **Frère adjacent :** Sélectionne le premier élément directement après le <h2>, ayant le même parent.
  h2 + p { ... }
- **Frères suivants :** Sélectionne tous les élément après le <h2>, ayant le même parent.
  h2 ~ p { ... }

> Usage

Cibler un élément selon sa position dans la structure HTML, sans ajouter de classes inutiles.

4. Sélecteurs et spécificités

4.4. **Sélecteurs structurels** (basés sur les relations dans le DOM)

> Spécificités :

- **Dépend de ce qui compose le sélecteur**
  Par exemple :
  - nav a = type + type → faible
  - .menu > li = classe + type → moyenne
  - #header nav a = identifiant + type + type → très élevée

- Ces sélecteurs ne "créent pas" de spécificité : ils additionnent celle des sélecteurs qu'ils contiennent.

4. Sélecteurs et spécificités

4.5. **Sélecteurs d'attribut** (basés sur les relations dans le DOM)

```
input[type="email"] { ... }
a[target="_blank"] { ... }
```

> Rôle

Ciblent un élément selon un attribut ou une valeur d'attribut.

> Usage

Différencier les champs de formulaires, les liens externes, ou des éléments enrichis de données (data-attributes).

> Spécificité : Équivalente à une classe

Ils sont donc plus forts qu'un type, moins forts qu'un identifiant.

4. Sélecteurs et spécificités

4.6. **Sélecteurs d'état** (pseudo-classes)

```
a:hover { ... }
input:focus { ... }
li:first-child { ... }
```

> Rôle

Ciblent un élément dans un état particulier : interaction, position, validation…

> Usage

Styles interactifs, accessibilité, comportements visuels : survol, focus, validité, ordre dans la liste, etc.

> Spécificité : Équivalente à une classe

Comme les sélecteurs d'attributs, les pseudo-classes sont considérées au même niveau.

4. Sélecteurs et spécificités

4.7. Pseudo-classes avancées

- Les pseudo-classes avancées permettent de cibler des états plus précis, des conditions logiques ou des positions complexes dans la structure HTML.
  Elles permettent de styliser des formulaires, des composants interactifs, ou des éléments dynamiques sans ajouter de classes supplémentaires.

4.7.1. Interaction et accessibilité

- améliore l'accessibilité
- évite les halos indésirables à la souris (focus-visible)

```
button:active        /* pendant le clic */
button:focus-visible  /* focus seulement au clavier */
```

4.7.2. Formulaires : état et validation

- indispensable pour styliser les formulaires modernes
- permet de gérer appels d'erreur, champs cochés, champs invalides…

```
input:required
input:valid
input:invalid
input:checked
input:disabled
```

4. Sélecteurs et spécificités

4.7. Pseudo-classes avancées

4.7.3. Pseudo-classes structurelles avancées

- ciblage ultra-précis
- alternance des couleurs (odd/even)
- sélecteurs conditionnels (:not())

```
li:nth-child(odd)
li:nth-child(2)
p:nth-of-type(3)
p:not(.intro)
```

4.7.4. Spécificité : Identique à une classe

4. Sélecteurs et spécificités

4.8. Pseudo-éléments

- Les pseudo-éléments permettent d'ajouter du contenu ou des éléments décoratifs **sans modifier le HTML**.
- Ils sont très utilisés pour enrichir l'interface (icônes, étiquettes, ornements, effets graphiques).

4.8.1. ::before et ::after

- ::before ajoute un élément **avant** le contenu réel
- ::after ajoute un élément **après**

> Nécessitent content (même vide : content: "")

```
button::before {
  content: "→ ";
}
```

- **Usages typiques :**
  - icônes décoratives
  - badges / pastilles
  - éléments graphiques (traits, formes)
  - enrichissement visuel sans surcharge HTML

4. Sélecteurs et spécificités

4.8. Pseudo-éléments

4.8.2. ::first-letter

- Styliser la première lettre d'un paragraphe.

```
p::first-letter {
  font-size: 2rem;
  font-weight: bold;
}
```

4.8.3. ::selection

- Styliser la sélection de texte.

```
::selection {
  background: #ffeb3b;
}
```

4.8.4. Pourquoi ils sont importants

- Permettent des décorations légères sans surcharge HTML
- Très utilisés en UI moderne (pastilles, icônes, surlignages)
- Essentiels en design system (éléments visuels récurrents)
- Conviennent parfaitement aux composants stylés avec Flexbox / Grid
- Compatibles avec les transitions et animations

4. Sélecteurs et spécificités

4.9. Cas particuliers

4.9.1. **Styles inline** (dans la balise html)

```
<p style="color:#333;">
```

> Spécificité : Très élevée

Ils écrasent presque tout. Déconseillé pour la maintenabilité.

4.9.2. !important

```
.alert {
    color: #F00 !important;
}
```

> Spécificité : Priorité absolue

Passe au-dessus de tous les autres sélecteurs.

Doit rester exceptionnel.

5. Parent / enfant / ancêtre (structure du DOM)

- Le CSS s'applique sur la structure offerte par le HTML.
  Pour comprendre comment cibler correctement un élément, il faut comprendre **comment les éléments HTML sont organisés entre eux**.
  Le DOM (Document Object Model) représente cette structure comme un **arbre**, où chaque élément occupe une position précise et entretient une relation avec les autres.

- **L'arbre du DOM : une organisation hiérarchique**
  Chaque élément HTML est placé dans une hiérarchie comparable à un organigramme.

  > Dans cette structure :
  - <body> est **l'ancêtre** de tout le reste
  - <section> est **le parent** de <h2> et <p>
  - <h2> et <p> sont **frères**, car ils ont le même parent
  - <h2> est **descendant** de <body>
  - <section> est **enfant** de <body>

```
<body>
   <section>
      <h2>Titre</h2>
      <p>Texte…</p>
   </section>
</body>
```

> Le DOM définit des **relations logiques**, indépendantes du style, qui servent de base au ciblage CSS.

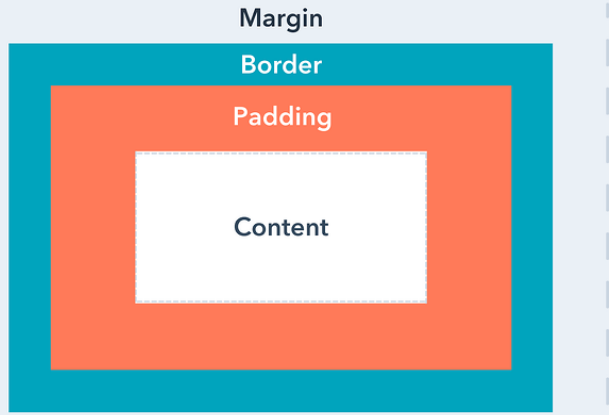
6. Le Box model

- Le **Box Model** est l'un des concepts centraux du CSS : il décrit la manière dont chaque élément HTML occupe de l'espace sur la page.
- Pour le navigateur, **tout élément est une boîte**, composée de plusieurs couches qui influencent sa taille, son positionnement et ses distances avec les autres éléments.
- Comprendre le Box Model est indispensable pour maîtriser la mise en page, gérer les espacements, éviter les décalages et construire des interfaces cohérentes.

6.1. Les quatre couches du Box Model

Chaque boîte est composée de **4 niveaux** :

1. **Content :** La zone où s'affiche le contenu (texte, image…).
2. **Padding :** Espace **interne** autour du contenu.
3. **Border :** Une bordure autour de l'élément.
4. **Margin :** Zone **externe**, entre l'élément et ceux qui l'entourent.



6. Le Box model

6.2. Le comportement naturel du Box Model

- Par défaut (modèle standard), la largeur totale d'un élément est calculée comme :
  Largeur totale = width + padding + border + margin

```
.box {
  width: 200px;
  padding: 20px;
  border: 2px solid;
  margin: 10px;
}
```

> Largeur totale = **10 + 2 + 20 + 200 + 20 + 2 + 10 = 264 px**

- Ce comportement peut rapidement rendre les mises en page complexes ou imprévisibles.

6. Le Box model

6.3. box-sizing: border-box : le réglage incontournable

- Pour simplifier les calculs et rendre les layouts cohérents, on utilise en RESET :

```
* {
  box-sizing: border-box;
}
```

- Avec border-box :
  Largeur totale = width (tout compris)

  - padding et border sont **inclus** dans la largeur définie,
  - l'élément devient beaucoup plus facile à dimensionner,
  - la mise en page est plus stable.

6. Le Box model

6.4. Les marges et la "margin collapse"

- Certaines marges verticales se **combinent** au lieu de s'additionner.
  C'est ce qu'on appelle la **collapse des marges** (fusion des marges).

- Exemple :

  > La marge finale entre les deux paragraphes = **30px**, pas 50px.

- Principes :
  - ne concerne que les marges **verticales**,
  - se produit entre éléments frères ou entre parent/enfant dans certains cas,
  - n'affecte jamais les marges horizontales.

6. Le Box model

6.5. Propriétés essentielles du Box model

- **Width / Height :** Définissent la taille du contenu (hors padding et border si content-box).

- **Padding :** Ajoute de l'espace interne.
  padding: 20px; /* identique sur tous les côtés */
  padding: 10px 20px; /* vertical horizontal */
  padding: 5px 10px 15px 0; /* top right bottom left */

- **Border :** Définit l'épaisseur, le style et la couleur.
  border: 1px solid #333; /* épaisseur style couleur */
  border-radius: 8px; /* 4 coins arrondis */

- **Margin :** Gère l'espace externe.
  Même logique que padding.

7. Typographie & Couleur

- La typographie et les couleurs jouent un rôle essentiel dans la lisibilité, la hiérarchie visuelle et l'esthétique générale d'une interface.
  Le CSS permet de contrôler précisément ces aspects afin d'offrir une expérience claire, cohérente et accessible.

7.1. La typographie

La typographie influence la perception du contenu : rythme, lisibilité, hiérarchie, confort de lecture.
Le CSS fournit un ensemble de propriétés dédiées à la gestion des polices et de la structure du texte.

7.1.1. Choisir et charger une police

- **Familles de polices (font-family)**

  Une pile de polices est recommandée :
  - nom précis de la police,
  - ensuite police de fallback compatible,
  - pour finir famille générique (serif, sans-serif, monospace).

```
body {
  font-family: "Inter", Arial, sans-serif;
}
```

7. Typographie & Couleur

7.1. La typographie

7.1.1. Choisir et charger une police

- **Polices importées (Google Fonts, etc.)**

```
@import url('https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap');
```

- **Conseils UX / accessibilité**
  - éviter les polices trop fines,
  - garantir un contraste suffisant,
  - utiliser un interlignage généreux.

7. Typographie & Couleur

7.1. La typographie

7.1.2. Taille du texte

Unités recommandées :

- **rem** : proportionnel à la taille de la racine (html), favorise l'accessibilité,
- **em** : proportionnel au parent, utile dans des composants indépendants.

```
body { font-size: 1rem; }
h1 { font-size: 2rem; }
```

7.1.3. Interlignage et espacement

Principes :

- un interlignage entre 1.4 et 1.8 améliore nettement la lisibilité,
- un espacement léger des lettres peut rendre un titre plus aéré.

```
p {
  line-height: 1.6;   /* interlignage */
  letter-spacing: 0.5px;
  word-spacing: 2px;
}
```

7. Typographie & Couleur

7.1. La typographie

7.1.4. Styles et Graisses

Options utiles :

- font-weight: 100 → 900 (de ultra-fin à ultra-gras)
- text-transform: uppercase, lowercase, capitalize
- text-decoration: underline, line-through

```
h2 {
  font-weight: 700;   /* graisse */
  font-style: italic;
  text-transform: uppercase;
}
```

7.1.5. Alignement et disposition du texte

Alignements possibles :

- left, right, center, justify.

```
p {
  text-align: left;
  text-indent: 20px;   /* retrait en
                  début de paragraphe */
}
```

7. Typographie & Couleur

7.2. Les couleurs

- Les couleurs contribuent à la hiérarchie visuelle, à l'esthétique et à l'accessibilité.
  Le CSS offre plusieurs façons de les définir.

7.2.1. Définir une couleur

Formats disponibles

- Hexadécimal : #ffffff
- RGB : rgb(255, 0, 0)
- RGBA (avec transparence) : rgba(0, 0, 0, 0.5)
- HSL (teinte, saturation, luminosité) : hsl(210, 70%, 50%)
- Variables CSS : var(--primary-color)

```
color: #333;
background-color: rgba(250, 250, 250, 0.5);
border-color: var(--primary-color)
```

7. Typographie & Couleur

7.2. Les couleurs

7.2.2. Gestion de la couleur de fond

- Le CSS permet de combiner plusieurs couches :
  Fond uni, dégradé, image.

```
#hero {
  background: #f4f4f4;
  background-image: url("pattern.png");
  background-repeat: no-repeat;
  background-size: cover;
}
```

7.2.3. Contraste et accessibilité

- Un bon contraste est essentiel pour le confort de lecture.
  Les outils comme les WCAG recommandent un ratio minimal de :
  - **4.5:1** pour le texte normal,
  - **3:1** pour les titres ou texte large.
- Conseils :
  - éviter les gris trop clairs sur fond clair,
  - garantir la lisibilité même en luminosité réduite,
  - tester les combinaisons avec un outil de contraste.

7. Typographie & Couleur

7.2. Les couleurs

7.2.4. Variables de couleur (Custom Properties)

- Pour améliorer la cohérence et faciliter les changements de thème :
- Avantages :
  - cohérence graphique,
  - changement facile (thème clair / sombre),
  - simplification de la maintenance.

```
:root {
  --color-primary: #185adb;
  --color-secondary: #f05454;
}

button {
  background: var(--color-primary);
}
```

7. Typographie & Couleur

7.3. Pourquoi la typographie et les couleurs sont essentielles ?

- Elles déterminent la lisibilité et la hiérarchie du contenu.
- Elles orientent le regard de l'utilisateur.
- Elles portent l'identité visuelle d'un projet.
- Elles garantissent l'accessibilité et le confort de lecture.
- Elles influencent directement la qualité perçue d'une interface.

8. Variables avancées / Design Tokens

- **Variables CSS : vers un design cohérent et modulable**

- Les variables CSS (Custom Properties) permettent de définir des valeurs réutilisables : couleurs, espacements, tailles de texte, radius…
  Elles facilitent la maintenance, l'uniformité visuelle et la création de thèmes (clair/sombre).

8.1. Déclarer des variables : la zone globale :root

- :root = équivalent du <html> → portée globale
- Toutes les valeurs deviennent accessibles dans la feuille de style entière

```
:root {
  --color-primary: #185adb;
  --color-secondary: #f05454;
  --font-base: 1rem;
  --spacing-sm: 0.5rem;
  --spacing-md: 1rem;
  --radius-sm: 4px;
  --radius-lg: 12px;
}
```

8.2. Utiliser une variable

- var() remplace directement une valeur CSS.

```
button {
  background: var(--color-primary);
  padding: var(--spacing-md);
  border-radius: var(--radius-sm);
}
```

8. Variables avancées / Design Tokens

8.3. Valeur de secours (fallback)

> si --text n'existe pas, #333 est utilisé.

```
p {
  color: var(--text, #333);
}
```

8.4. Design tokens : structurer les variables

- Les **design tokens** sont un **concept métier**, issu des *design systems*. Ils représentent les **valeurs clés de l'identité visuelle** :
  - couleurs de marque
  - tailles typographiques
  - espacements
  - rayons d'arrondis
  - ombres
  - durées d'animation

> Ce sont des **décisions de design** mises sous forme de valeurs.

```
:root {
  /* Couleurs */
  --color-bg: #ffffff;
  --color-text: #111;
  --color-accent: #0066ff;

  /* Typographie */
  --font-size-base: 1rem;
  --font-size-lg: 1.25rem;

  /* Espacements */
  --space-1: 0.5rem;
  --space-2: 1rem;

  /* Rayons */
  --radius-base: 8px;
}
```

8. Variables avancées / Design Tokens

8.5. Pourquoi utiliser les variables ?

- centraliser les couleurs, tailles, espacements
- changer un thème en une seule instruction
- rendre le design cohérent
- faciliter la maintenance du CSS
- éviter les répétitions (DRY)
- indispensable dans les systèmes actuels (Design Systems, UI kits)

9. Positionnement CSS : contrôler la place d'un élément dans la page

- Le module *position* permet de retirer un élément du flux normal, de le fixer, de le superposer ou de le coller lors du défilement.
  > Indispensable pour créer des menus, badges, pop-ups, overlays, tooltips, sticky headers, etc.

9.1. position: static (par défaut)

```
.box { position: static; }
```

- élément dans le flux normal
- **top / left / bottom / right** n'ont aucun effet
- aucun positionnement manuel

9.2. position: relative

```
.box {
    position: relative;
    top: 10px;
    left: 20px;
}
```

- reste dans le flux normal
- peut être **décalé sans affecter les autres éléments**
- sert souvent de **référence** pour un élément enfant en *absolute*

9. Positionnement CSS : contrôler la place d'un élément dans la page

9.3. position: absolute

- retiré du flux normal
- se positionne **par rapport au premier parent positionné** (non-static)
- permet les badges, pop-ups internes, éléments superposés

```
.badge {
  position: absolute;
  top: 0;
  right: 0;
}
```

9.4. position: fixed

- retiré du flux
- **fixé à l'écran**, même lors du défilement
- utile pour un header collé, un bouton retour en haut, une barre latérale fixe…

```
header {
  position: fixed;
  top: 0;
  width: 100%;
}
```

9. Positionnement CSS : contrôler la place d'un élément dans la page

9.5. position: sticky

- hybride entre *relative* et *fixed*
- reste dans le flux, puis devient fixe quand on dépasse un seuil
- idéal pour : menus, titres de sections, sommaires

```
nav {
  position: sticky;
  top: 10px;
}
```

9.6. superposition : z-index

- contrôle l'ordre d'empilement
- ne fonctionne que sur des éléments **positionnés** (relative, absolute, fixed, sticky)

```
.modal {
  position: fixed;
  z-index: 1000;
}
```

9. Positionnement CSS : contrôler la place d'un élément dans la page

9.7. Rôle du positionnement CSS

- créer des composants superposés (modales, tooltips, overlays)
- aligner ou isoler un élément dans une zone précise
- produire des interfaces interactives (menus collants, pop-ups)
- contrôler la hiérarchie visuelle en 3D (z-index)

10. Transitions & Animations

10.1. Transitions : animer un changement d'état

- Les transitions permettent d'ajouter une animation fluide lorsqu'une propriété change : survol, focus, ouverture, fermeture, interaction…

10.1.1. Syntaxe de base

- all : propriétés animées
- 0.3s : durée
- ease : courbe d'accélération

```
.button {
  transition: all 0.3s ease;
}
```

10.1.2. Exemple simple

- agrandissement doux au survol.

```
.card {
  transition: transform 0.3s ease;
}
.card:hover {
  transform: scale(1.05);
}
```

10.1.3. Détails utiles

- Propriétés animables : couleur, background, transform, opacity, border-radius…
- Transition **ne fonctionne que si une propriété change**
- Éviter d'animer width / height (coûts élevés), privilégier transform

10. Transitions & Animations

10.2. Animations : contrôler la progression dans le temps

- Les animations permettent d'animer une propriété indépendamment d'un état (boucle, délai, trajectoire…).

10.2.1. @keyframes : définir une animation

- On définit fadeIn

```
@keyframes fadeIn {
  from { opacity: 0; }
  to   { opacity: 1; }
}
```

10.2.2. Application

- On applique fadeIn

```
.modal {
  animation: fadeIn 0.6s ease-out;
}
```

10. Transitions & Animations

10.3. Les courbes d'animation

Fournissent un effet plus naturel.

- ease : standard, fluide
- ease-in : démarre lentement
- ease-out : finit doucement
- ease-in-out : doux au début et à la fin
- linear : vitesse constante
- cubic-bezier(…) : contrôle total de la courbe

10.4. Rôle des transitions & animations

- rendre l'interface plus fluide
- guider l'attention (feedback visuel)
- améliorer la perception d'instantanéité
- renforcer l'interaction sans recourir au JavaScript
- respecter l'accessibilité grâce aux media queries adaptées

11. Effets visuels en CSS : enrichir l'apparence d'un élément

- Le CSS permet d'ajouter des effets graphiques légers et performants pour améliorer la lisibilité, la profondeur et l'esthétique de l'interface.
  Ombres, arrondis, filtres, dégradés : des outils essentiels pour les composants.

11.1. Ombres (box-shadow / text-shadow)

- **box-shadow**
  - ombre externe (effet de profondeur)
  - paramètres : x, y, flou, étendue (optionnelle), couleur
  - fonctionne sur n'importe quel élément

```
.card {
  box-shadow: 0 4px 12px rgba(0,0,0,0.15);
}
```

- **text-shadow**
  - ombre portée sur du texte
  - à utiliser avec modération (lisibilité)

```
h1 {
  text-shadow: 2px 2px 4px rgba(0,0,0,0.3);
}
```

11. Effets visuels en CSS : enrichir l'apparence d'un élément

11.2. Arrondis : border-radius

Exemples fréquents :

- border-radius: 50%; → cercle
- border-radius: 0 12px 12px 0; → arrondis asymétriques

```
.button {
  border-radius: 9px;
}
```

11.3. Dégradés : linear-gradient, radial-gradient

- **Linear**

```
#hero {
  background: linear-gradient(135deg, #4A90E2, #9013FE);
}
```

- **Radial**

```
.box {
  background: radial-gradient(circle, #fff, #ccc);
}
```

> très utilisés pour des en-têtes, fonds de cartes, transitions douces

> se comportent comme une **image de fond CSS**

11. Effets visuels en CSS : enrichir l'apparence d'un élément

11.4. Filtres (filter) et flous d'arrière-plan (backdrop-filter)

- **filter** (sur l'élément)
  - effets : blur, brightness, contrast, grayscale, sepia, drop-shadow

  > utiles pour : hover sur images, miniatures, retouches légères

```
img {
  filter: grayscale(80%);
}
```

- **backdrop-filter** (sur l'arrière-plan)
  - crée des effets "verre dépoli" (glassmorphism)

  > nécessite un fond semi-transparent

```
.panel {
  backdrop-filter: blur(12px);
}
```

11.5. Arrière-plans avancés

- **Image + couleur**
  - superposition possible (image + gradient)

  > utile pour lisibilité sur photos

```
header {
  background: url(banner.jpg) center/cover no-repeat,
              linear-gradient(#0008, #0008);
}
```

11. Effets visuels en CSS : enrichir l'apparence d'un élément

11.6. Quand utiliser ces effets ?

- créer une hiérarchie visuelle (ombrage léger sur cartes)
- séparer des sections (dégradés ou arrière-plans semi-transparents)
- renforcer l'identité visuelle (angles arrondis, ombres douces)
- améliorer la lisibilité (overlay, blur de fond)
- ajouter du relief sans images lourdes

12. Flexbox & Grid

- Les systèmes modernes de mise en page du CSS reposent essentiellement sur deux modules puissants : **Flexbox** et **Grid**.
  Ils remplacent avantageusement les anciennes méthodes (float, tableaux, positionnement manuel) et permettent de construire des interfaces adaptatives, stables et cohérentes.

12.1. Flexbox : organiser les éléments en 1 dimension

- Flexbox (Flexible Box Layout) permet de disposer les éléments **soit en ligne**, soit **en colonne**, tout en gérant facilement l'espace, les alignements et les proportions.

- Il est conçu pour organiser **une dimension à la fois** :
  - une **ligne** (row)
  - ou une **colonne** (column)

12. Flexbox & Grid

12.1. Flexbox : organiser les éléments en 1 dimension

12.1.1. Activer Flexbox

- Dès qu'un conteneur passe en display: flex, ses enfants deviennent des **flex items** contrôlables.

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

12.1.2. Orientation

- Flex-direction gère l'orientation.

```
flex-direction: row;     /* par défaut : horizontal */
flex-direction: column;  /* vertical */
```

12.1.3. Axe principal (main axis horizontal)

- justify-content gère la répartition **le long de l'axe principal**.

```
justify-content: flex-start;
justify-content: center;
justify-content: space-between;
justify-content: space-around;
```

12.1.4. Axe secondaire (cross axis vertical)

- align-items gère l'alignement **perpendiculaire**.

```
align-items: flex-start;
align-items: center;
align-items: flex-end;
```

12. Flexbox & Grid

12.1. Flexbox : organiser les éléments en 1 dimension

12.1.5. Gestion de l'espace

- **Croissance / rétrécissement / taille de base**
  Contrôle de chaque item individuellement :

```
.item {
  flex-grow: 1;    /* occupe l'espace disponible */
  flex-shrink: 1;  /* rétrécit si nécessaire */
  flex-basis: 200px; /* taille de base */
}
Raccourci en
.item {
  flex: 1; /* équivalent à grow:1 shrink:1 basis:0 */
}
```

12.1.6. Retour à la ligne

- Permet de créer des lignes multiples (pratique pour les galeries d'images ou les cartes d'articles).

```
flex-wrap: wrap;
```

12. Flexbox & Grid

12.1. Flexbox : organiser les éléments en 1 dimension

12.1.7. Quand utiliser Flexbox ?

- Barre de navigation
- Listes horizontales
- Cartes alignées
- Centrage vertical / horizontal
- Composants autonomes (boutons, contrôles, formulaires)

> Flexbox = un axe principal.

12. Flexbox & Grid

12.2. CSS Grid : mise en page en 2 dimensions

- CSS Grid permet de structurer des layouts complexes en combinant **colonnes et lignes**.
  Il est pensé pour la construction de **grilles**, de gabarits, de pages complètes ou de blocs structurés.
- **Grid = 2 axes (colonnes + lignes).**

12.2.1. Activer Grid

- Dès qu'un conteneur passe en display: grid,
  ses enfants deviennent contrôlables.

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

12.2.2. Définir des colonnes et des lignes

- **Unités utiles :**
  - fr : fraction d'espace disponible (la + pratique)
  - %, px, minmax()

```
grid-template-columns: 1fr 2fr 1fr; /* 3 colonnes */
grid-template-rows: auto 200px auto;  /* 3 lignes */
```

12. Flexbox & Grid

12.2. CSS Grid : mise en page en 2 dimensions

12.2.3. Espacement

```
gap: 20px;     /* espace entre colonnes ET lignes */
column-gap: 10px;
row-gap: 30px;
```

12.2.4. Positionnement d'un élément dans la grille

- Possibilité d'étendre un item sur plusieurs lignes ou colonnes.

```
.item {
  grid-column: 1 / 3;    /* de la colonne 1 à la 3 */
  grid-row: 2 / 4;       /* de la ligne 2 à la 4 */
}
```

12.2.5. Grilles réactives

- Grâce aux fonctions avancées :
  → La grille s'adapte automatiquement à la largeur de l'écran.

```
grid-template-columns: repeat(auto-fit,
                                minmax(250px, 1fr));
```

12. Flexbox & Grid

12.2. CSS Grid : mise en page en 2 dimensions

12.2.6. Quand utiliser Grid ?

- Galeries,
- Portfolios,
- Magazines…

> Grid = structure globale. Flexbox = alignement local.

12.3. Combiner Flexbox et Grid

- Dans les projets, on utilise très souvent les deux :
  - **Grid** pour structurer les galeries et cartes d'une page.
  - **Flexbox** pour organiser l'intérieur d'un composant (menu, bouton, formulaire…).
- Exemple classique :
  - Flexbox dans chaque carte d'article et pour la navigation
  - Grid pour la galerie de contenu
- Ils ne s'opposent pas : **ils se complètent.**

13. Responsive design

- Le responsive design consiste à adapter l'affichage d'un site web aux différents types d'écrans : mobile, tablette, ordinateur, très grand écran…
- L'objectif est d'assurer une **lecture confortable**, une **navigation fluide** et une **cohérence visuelle**, quelle que soit la taille du support.
- Le CSS fournit plusieurs outils pour adapter les layouts, ajuster les composants et optimiser l'expérience sur l'ensemble des appareils.

13. Responsive design

13.1. Les media queries : le cœur du responsive

- Les **media queries** permettent d'exécuter du CSS seulement si certaines conditions sont remplies : largeur, hauteur, orientation…

13.1.1. Exemple simple

> Quand l'écran fait 768px ou moins,
> les éléments possédant une classe flex,
> passent en colonne, ils ne sont plus flexibles
> et se comportent comme des blocs.

```
@media (max-width: 768px) {
  .flex {
    display: block;
  }
}
```

13. Responsive design

13.1. Les media queries : le cœur du responsive

13.1.2. Breakpoints courants (références usuelles)

- Ces valeurs ne sont pas des règles strictes, mais des points d'adaptation fréquents :
  - **≥ 1200px** : grands écrans
  - **992px – 1199px** : desktop
  - **768px – 991px** : tablettes
  - **480px – 767px** : mobiles larges
  - **≤ 479px** : petits mobiles

- Le plus important n'est pas la valeur, mais **l'intention** :
  adapter le contenu quand il devient difficile à lire.

13. Responsive design

13.1. Les media queries : le cœur du responsive

13.1.3. Mobile-first vs Desktop-first

- **Mobile-first (recommandé aujourd'hui)**
  On commence avec des styles pour petits écrans, puis on élargit :

  Avantages :
  - plus léger,
  - plus logique pour la majorité des usages,
  - assure une bonne lisibilité dès les petits écrans.

- **Desktop-first**
  À éviter pour les nouveaux projets, sauf cas particulier.

```
/* mobile par défaut */
.container { flex-direction: column; }

/* desktop ensuite */
@media (min-width: 768px) {
  .container { flex-direction: row; }
}
```

13. Responsive design

13.2. Images et médias responsives

13.2.1. Images flexibles

- L'image s'adapte à son conteneur.

```
CSS
img {
  max-width: 100%;
  height: auto;
}
```

13.2.2. Formats multiples

- Pour fournir plusieurs versions d'une image selon la résolution :

```
HTML
<img
  src="image-800.jpg"
  srcset="image-400.jpg 400w, image-800.jpg 800w,
image-1200.jpg 1200w"
  sizes="(max-width: 600px) 100vw, 50vw"
/>
```

13. Responsive design

13.3. Flexbox et Grid au service du responsive

- Les deux systèmes s'adaptent naturellement à la taille de l'écran.

- **Exemple Grid responsive :**
  La grille se réorganise automatiquement :
  nombre de colonnes variable selon la largeur.

```
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fit,
minmax(220px, 1fr));
  gap: 20px;
}
```

- **Exemple Flexbox responsive :**
  Les éléments passent automatiquement
  à la ligne lorsque l'espace est insuffisant.

```
.cards {
  display: flex;
  flex-wrap: wrap;
  gap: 20px;
}
```

13. Responsive design

13.4. Optimisation de l'expérience sur mobile

- **Taille des zones cliquables**
  > Les recommandations courantes préconisent un minimum d'environ **44px** d'espace cliquable pour les boutons.

- **Espace autour du texte**
  > L'interlignage doit rester confortable (1.4–1.8) même sur petits écrans.

- **Éviter les textes trop longs**
  > Les colonnes de texte trop larges ou trop serrées fatiguent la lecture.

- **Eviter les images trop lourdes**
  > Images adaptées au mobile pour réduire le temps de chargement.

- **Utiliser des menus adaptatifs :** - hamburger menu
  - menu déroulant
  - liste verticale

13. Responsive design

13.5. Pourquoi le responsive est indispensable ?

- La majorité du trafic web est aujourd'hui mobile.
- Les moteurs de recherche privilégient les sites adaptés aux petits écrans.
- L'absence d'adaptation rend la lecture difficile.
- Un site responsive améliore la satisfaction et l'efficacité de l'utilisateur.
- Il garantit une expérience cohérente sur tous les supports.