Cours - CSS Avancées (transitions, transformations et variables CSS)

> **Note importante : Ce document est conçu comme un guide de référence, mais n'est pas exhaustif. La capacité à rechercher des informations complémentaires et à résoudre des problèmes de manière autonome est une compétence essentielle pour tout développeur. Si vous rencontrez des difficultés ou si certaines étapes ne fonctionnent pas comme prévu, n'hésitez pas à consulter d'autres ressources (documentation officielle, tutoriels YouTube, forums spécialisés comme Stack Overflow). Apprendre à trouver l'information manquante et à débugger par vous-même est probablement la compétence la plus précieuse que vous développerez**

Introduction aux transitions et transformations CSS

W3Schools.com

W3Schools offers free online tutorials, references and exercises in all the major languages of the web. Covering popular subjects like HTML, CSS, JavaScript, Python, SQL,

https://www.w3schools.com/css/css3_transitions.asp

💡 Les transitions et transformations CSS sont des outils puissants qui permettent d'ajouter des animations et des effets visuels aux éléments HTML sans avoir besoin de JavaScript. Ces fonctionnalités permettent de créer des interfaces utilisateur dynamiques et interactives, améliorant ainsi l'expérience utilisateur avec des micro-interactions fluides et élégantes.

Les propriétés de transition

Les transitions CSS permettent de modifier graduellement les valeurs d'une propriété sur une période définie, créant ainsi des animations fluides entre deux états.

Propriétés de base des transitions

```css
.element {
  /* Propriété(s) à animer */
  transition-property: background-color, transform;

  /* Durée de l'animation (en secondes ou millisecondes) */
  transition-duration: 0.3s;

  /* Courbe d'accélération */
  transition-timing-function: ease-in-out;

  /* Délai avant le démarrage de l'animation */
  transition-delay: 0.1s;

  /* Notation condensée: property duration timing-function delay */
  transition: background-color 0.3s ease-in-out 0.1s, transform 0.5s ease;
}
```

- `transition-property` : spécifie les noms des propriétés CSS auxquelles un effet de transition doit être appliqué
- `transition-duration` : définit la durée nécessaire pour qu'une transition s'effectue complètement

- `transition-timing-function` : définit comment les valeurs intermédiaires des propriétés affectées par une transition sont calculées
- `transition-delay` : spécifie le moment où l'effet de transition doit commencer

Courbes d'accélération (timing functions)

```css
.element {
  /* Différentes valeurs pour transition-timing-function */

  /* Accélération progressive */
  transition-timing-function: ease-in;

  /* Décélération progressive */
  transition-timing-function: ease-out;

  /* Combinaison des deux */
  transition-timing-function: ease-in-out;

  /* Vitesse constante */
  transition-timing-function: linear;

  /* Courbe de Bézier personnalisée */
  transition-timing-function: cubic-bezier(0.68, -0.55, 0.27, 1.55);

  /* Effet de ressort */
  transition-timing-function: cubic-bezier(0.175, 0.885, 0.32, 1.275);
}
```

▼ **Informations complémentaires (cubic-bezier)**

La fonction `cubic-bezier()` est une fonction de temporisation personnalisée qui permet de définir précisément le rythme d'une animation CSS.

- `cubic-bezier(x1, y1, x2, y2)` où les valeurs représentent les points de contrôle d'une courbe de Bézier cubique.
- Les quatre valeurs sont des coordonnées des deux points de contrôle (P1 et P2) de la courbe, avec x1, y1 pour P1 et x2, y2 pour P2. Les points P0(0,0) et P3(1,1) sont fixes.

- Les valeurs x sont normalement entre 0 et 1, mais les valeurs y peuvent dépasser cette plage pour créer des effets de rebond.
- `cubic-bezier(0.68, -0.55, 0.27, 1.55)` crée un effet de rebond car y1 est négatif (-0.55) et y2 dépasse 1 (1.55).
- CSS propose des mots-clés comme `ease`, `linear`, `ease-in`, `ease-out` et `ease-in-out` qui sont en fait des raccourcis pour des valeurs cubic-bezier spécifiques.
- La courbe représente la progression de l'animation dans le temps, où l'axe X représente la durée et l'axe Y représente la progression.
- Il existe des outils interactifs pour créer et visualiser des courbes de Bézier (comme cubic-bezier.com).
- Idéal pour créer des animations naturelles, des effets de rebond, d'élasticité ou d'anticipation.
- Les animations qui utilisent cubic-bezier sont généralement bien optimisées par les navigateurs modernes.

Utilisation avec les états hover, focus, active

```
.bouton {
  background-color: #3498db;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.bouton:hover {
  background-color: #2980b9;
  transform: scale(1.05);
}

.bouton:active {
  transform: scale(0.98);
}
```

```css
.champ-texte {
  border: 2px solid #ccc;
  padding: 8px;
  transition: border-color 0.3s ease;
}

.champ-texte:focus {
  border-color: #3498db;
  outline: none;
}
```

Les custom properties (variables CSS)

Les custom properties (aussi appelées variables CSS) permettent de stocker des valeurs qui peuvent être réutilisées dans votre feuille de style. Elles sont particulièrement utiles pour gérer les couleurs, les durées de transition et autres valeurs récurrentes.

Déclaration et utilisation des variables CSS

```css
/* Déclaration au niveau de la racine pour une portée globale */
:root {
  --couleur-primaire: #3498db;
  --couleur-secondaire: #2980b9;
  --duree-transition: 0.3s;
  --effet-transition: ease-in-out;
  --echelle-hover: 1.05;
}

.bouton {
  background-color: var(--couleur-primaire);
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  transition: background-color var(--duree-transition) var(--effet-transitio
n),
              transform calc(var(--duree-transition) - 0.1s) var(--effet-transitio
```

```
n);
}

.bouton:hover {
  background-color: var(--couleur-secondaire);
  transform: scale(var(--echelle-hover));
}
```

- `var()` : fonction qui permet d'insérer la valeur d'une custom property
- `:root` : pseudo-classe qui correspond à l'élément racine du document (généralement `<html>`)

Portée et héritage des variables CSS

```
:root {
  --couleur-texte: #333;
}

.conteneur {
  --couleur-texte: #666; /* Redéfinition locale */
}

.conteneur p {
  color: var(--couleur-texte); /* Utilisera #666 */
}

body > p {
  color: var(--couleur-texte); /* Utilisera #333 */
}
```

Variables CSS avec valeur de repli

```
.element {
  /* Utilise la variable si elle existe, sinon utilise la valeur de repli */
  color: var(--couleur-non-definie, #000);

  /* Chaînage de variables avec repli */
```

```
  margin: var(--marge-personnalisee, var(--marge-standard, 16px));
}
```

Variables CSS et states (hover, media queries)

```
.bouton {
  --couleur-fond: #3498db;
  background-color: var(--couleur-fond);
  transition: background-color 0.3s;
}

.bouton:hover {
  --couleur-fond: #2980b9; /* La transition s'appliquera car la propriété de base reste la même */
  /* background-color hérite automatiquement de la nouvelle valeur de --couleur-fond */
}

@media (prefers-color-scheme: dark) {
  :root {
    --couleur-texte: #fff;
    --couleur-fond: #222;
  }
}
```

Les transformations CSS

Les transformations CSS permettent de modifier visuellement un élément en le déplaçant, le redimensionnant, le faisant pivoter ou le déformant dans l'espace 2D (ou 3D).

Translate (déplacement)

```
.element {
  /* Déplacement horizontal et vertical */
  transform: translate(20px, 30px);
```

```
  /* Déplacement horizontal uniquement */
  transform: translateX(20px);

  /* Déplacement vertical uniquement */
  transform: translateY(30px);

  /* Déplacement en pourcentage de la taille de l'élément */
  transform: translate(50%, -25%);
}
```

- `translate()` : déplace un élément horizontalement et/ou verticalement

Scale (redimensionnement)

```
.element {
  /* Redimensionnement horizontal et vertical */
  transform: scale(1.5, 2);

  /* Redimensionnement uniforme */
  transform: scale(1.5);

  /* Redimensionnement horizontal uniquement */
  transform: scaleX(1.5);

  /* Redimensionnement vertical uniquement */
  transform: scaleY(2);
}
```

- `scale()` : change la taille d'un élément

Rotate (rotation)

```
.element {
  /* Rotation en degrés (sens horaire) */
  transform: rotate(45deg);

  /* Rotation en degrés (sens anti-horaire) */
  transform: rotate(-90deg);
```

```
  /* Autres unités possibles: rad, grad, turn */
  transform: rotate(0.5turn); /* 180 degrés */
}
```

- `rotate()` : fait pivoter un élément autour d'un point fixe

Skew (inclinaison)

```
.element {
  /* Inclinaison horizontale et verticale */
  transform: skew(15deg, 10deg);

  /* Inclinaison horizontale uniquement */
  transform: skewX(15deg);

  /* Inclinaison verticale uniquement */
  transform: skewY(10deg);
}
```

- `skew()` : incline un élément dans une ou plusieurs directions

Combinaison de transformations

```
.element {
  /* Plusieurs transformations appliquées dans l'ordre spécifié */
  transform: translate(20px, 10px) rotate(45deg) scale(1.5);
}
```

Point d'origine des transformations

```
.element {
  /* Définit le point autour duquel la transformation est appliquée */
  transform-origin: center;
  transform-origin: top left;
  transform-origin: bottom right;
  transform-origin: 25px 50px;
```

```
  transform-origin: 50% 50%;
}
```

- `transform-origin` : définit l'origine des transformations d'un élément

Exemple pratique : Menu hamburger responsive avec variables CSS

```
:root {
  --couleur-menu: #333;
  --couleur-fond-menu: #fff;
  --duree-transition: 0.4s;
  --timing-function: ease-in-out;
  --largeur-menu: 280px;
  --epaisseur-barre: 3px;
  --espacement-barres: 6px;
}

.menu-hamburger {
  display: none;
  cursor: pointer;
}

.barre {
  width: 30px;
  height: var(--epaisseur-barre);
  background-color: var(--couleur-menu);
  margin: var(--espacement-barres) 0;
  transition: var(--duree-transition);
}

.menu-mobile {
  position: fixed;
  top: 0;
  right: calc(-1 * var(--largeur-menu)); /* Caché par défaut */
  width: var(--largeur-menu);
  height: 100%;
```

```
  background-color: var(--couleur-fond-menu);
  box-shadow: -2px 0 5px rgba(0, 0, 0, 0.2);
  transition: right var(--duree-transition) var(--timing-function);
}

.menu-mobile.ouvert {
  right: 0; /* Visible quand la classe .ouvert est appliquée */
}

/* Animation du menu hamburger */
.menu-hamburger.actif .barre:nth-child(1) {
  transform: rotate(-45deg) translate(-5px, 6px);
}

.menu-hamburger.actif .barre:nth-child(2) {
  opacity: 0;
}

.menu-hamburger.actif .barre:nth-child(3) {
  transform: rotate(45deg) translate(-5px, -6px);
}

@media (max-width: 768px) {
  .menu-desktop {
    display: none;
  }

  .menu-hamburger {
    display: block;
  }
}

/* Adaptation pour le mode sombre */
@media (prefers-color-scheme: dark) {
  :root {
    --couleur-menu: #fff;
    --couleur-fond-menu: #222;
```

```
  }
}
```

Le rôle de JavaScript dans les transitions et transformations

> **Note importante** : Cette section ne fais pas partie du cours, mais je l'ajoute pour répondre aux questions que vous pourriez vous poser. Elle est uniquement destinée aux étudiants qui se demanderaient pourquoi JavaScript est parfois utilisé avec les transitions CSS. Ne vous inquiétez pas si vous ne comprenez pas toutes les notions abordées ici, c'est normal et cela ne fait pas partie des compétences attendues à ce stade de votre formation.

Bien que CSS soit extrêmement puissant pour créer des animations et des transitions, JavaScript reste souvent nécessaire pour des interactions plus complexes, notamment dans les menus déroulants et autres composants interactifs.

Pourquoi combiner JavaScript avec CSS ?

Les développeurs ajoutent du JavaScript aux menus déroulants pour plusieurs raisons, même si techniquement on peut réaliser un menu basique avec du CSS uniquement :

1. **Accessibilité améliorée** : Le JavaScript permet de gérer correctement les attributs ARIA, la navigation au clavier, et le focus, rendant le menu plus accessible aux technologies d'assistance.
2. **Interactions complexes** : Pour des comportements plus sophistiqués comme des sous-menus à plusieurs niveaux, des fermetures conditionnelles, ou des animations contextuelles.
3. **États dynamiques** : Gestion d'états qui ne peuvent pas être contrôlés uniquement avec CSS, comme marquer l'élément actif du menu en fonction de la page courante.

4. **Événements avancés** : Pour détecter des interactions comme le clic en dehors du menu pour le fermer, le maintien prolongé sur un élément, ou la détection de glissement tactile.
5. **Compatibilité navigateur** : Certaines fonctionnalités CSS avancées ne sont pas supportées uniformément sur tous les navigateurs, donc JavaScript peut servir de solution de repli.
6. **Intégration avec le state de l'application** : Dans les applications modernes (React, Vue, etc.), le menu doit souvent refléter l'état global de l'application.
7. **Meilleures performances sur mobile** : Parfois, les interactions purement CSS peuvent être saccadées sur certains appareils mobiles.

Cela dit, pour un menu déroulant simple et standard, l'approche CSS avec media queries, transitions et transforms reste tout à fait valable et représente souvent la solution la plus légère et performante.

> **Rappel :** Ce contenu est simplement informatif et vous pouvez tout à fait créer des menus déroulants efficaces avec les transitions et transformations CSS que nous avons vues dans ce cours, sans JavaScript.

▼ **Exemple de combinaisons des deux**

```
/* Le CSS gère l'apparence et les transitions */
.menu-deroulant {
  transform: translateY(-20px);
  opacity: 0;
  visibility: hidden;
  transition: transform 0.3s var(--timing-function),
              opacity 0.3s var(--timing-function),
              visibility 0s 0.3s;
}

.menu-deroulant.actif {
  transform: translateY(0);
  opacity: 1;
  visibility: visible;
  transition: transform 0.3s var(--timing-function),
```

```
            opacity 0.3s var(--timing-function),
            visibility 0s;
}
```

```
// JavaScript gère les interactions et états
document.querySelector('.menu-toggle').addEventListener('click', funct
ion() {
  const menu = document.querySelector('.menu-deroulant');
  menu.classList.toggle('actif');

  // Gestion d'accessibilité
  const expanded = menu.classList.contains('actif');
  this.setAttribute('aria-expanded', expanded);
  menu.setAttribute('aria-hidden', !expanded);
});

// Fermeture du menu si clic à l'extérieur
document.addEventListener('click', function(event) {
  const menu = document.querySelector('.menu-deroulant');
  const toggle = document.querySelector('.menu-toggle');

  if (!menu.contains(event.target) && !toggle.contains(event.target) &&
menu.classList.contains('actif')) {
    menu.classList.remove('actif');
    toggle.setAttribute('aria-expanded', 'false');
    menu.setAttribute('aria-hidden', 'true');
  }
});
```

Bonnes pratiques

1. Privilégiez les transitions sur les propriétés peu coûteuses en termes de performance (opacity, transform) plutôt que margin, width ou height
2. Gardez les animations courtes (0.2s à 0.5s) pour une expérience utilisateur réactive
3. Utilisez `will-change` avec parcimonie pour optimiser les performances des animations complexes

4. Préférez `transform: translate()` à `position: absolute` pour les animations de déplacement
5. Utilisez les custom properties pour centraliser les valeurs importantes (couleurs, durées, etc.)
6. Combinez les custom properties avec `calc()` pour des calculs dynamiques
7. N'animez pas plus de 3 propriétés à la fois pour éviter les problèmes de performance
8. Testez vos animations sur différents appareils pour vérifier qu'elles restent fluides

---