Les animations CSS

Chapitre 1 : **Découvrez les animations web**

- vous pouvez créer vos **animations CSS** sur vos propres éditeurs de code, ou bien sur des outils de code en ligne tels que CodePen ou CodeSandbox ;
- l'animation a été démocratisée auprès du grand public par **Disney** ;
- les techniques d'animation web n'ont cessé d'évoluer depuis les débuts du web : **GIF**, **Flash**, etc. ;
- aujourd'hui, il existe de nombreuses techniques d'animation web : JavaScript, SVG, Canvas, WebGL mais surtout CSS ;
- l'animation permet d'**ajouter de la vie** à une page web, et ainsi de vraiment impacter l'expérience d'un utilisateur sur cette page.

Chapitre 2 : Créez des animations simples avec les transitions

Résumé :

les 5 éléments nécessaires pour créer une transition sont :

  - une propriété CSS à modifier,
  - une valeur initiale pour votre propriété CSS,
  - une valeur finale pour cette même propriété,
  - une durée,
  - une pseudoclasse pour déclencher l'animation ;

- on applique la valeur initiale à l'élément qu'on veut modifier, et la valeur finale dans la pseudoclasse qui déclenche la transition ;
- la durée peut s'exprimer en secondes : 0.4s, ou en millisecondes : 400ms ;
- les propriétés d'une transition peuvent être déclarées individuellement : transition-duration: 400ms ;
- ou bien combinées en une seule propriété comme : transition: transform 400ms .

Si votre transition ne se déclenche pas, regardez où vous avez placé vos **propriétés** de transition. Elles doivent être dans le sélecteur qui contient le point de départ de l'élément.

"Refactorer" veut dire *réviser le code* pour qu'il soit mieux écrit, plus efficace. Ce mot est un anglicisme à partir de *refactoring*.

3/ Déclenchez vos transitions avec les pseudoclasses

L'utilisation d'une pseudoclasse pour déclencher une transition est l'un des points essentiels des animations CSS. il existe aujourd'hui plus de 50 pseudoclasses.

Pour qu'une pseudoclasse déclenche une transition sur un autre élément, cet élément doit être le voisin suivant direct (ou *sibling* en anglais) dans le document HTML. De fait, nous devons utiliser le combinateur **d'adjacence +** pour créer la transition sur l'élément voisin.

Résumé :

- les pseudoclasses sont essentielles pour **déclencher** des transitions en CSS ;
- les pseudoclasses les plus adaptées pour déclencher des transitions sont celles qui impliquent une **interaction** avec l'utilisateur ;
- les pseudoclasses les plus courantes pour déclencher une transition sont `:hover`, `:active`, `:focus`, `:valid`, `:invalid`, `:not()`, `:checked`, `:enabled`, et `:disabled`
- on peut **combiner** des pseudoclasses entre elles pour créer des sélecteurs plus précis ;
- les pseudoclasses peuvent aussi être utilisés pour changer le style d'un élément voisin.

Chapitre 4 : Appliquez les 12 principes de l'animation au web

Les objets ont une masse, ils doivent accélérer et décélérer. Les mouvements, à une échelle ou une autre, tremblent, oscillent, changent de direction : nous sommes contraints par les lois de la physique. Ce que nos cerveaux trouvent naturels, ce sont des mouvements **irréguliers**. C'est pour cette raison que les valeurs par défaut de nos animations risquent de paraître lisses, artificielles ou robotiques.

Résumé :

1- **Squash and Stretch (compression et étirement) (**exemple :balle au caoutchou rebondissante**)**

2- **Anticipation :** permet de **préparer l'audience** (ici, l'utilisateur) à une action à venir…

3- Staging (mise en scène) **guider** les yeux des utilisateurs, vers les parties importantes de l'écran par l'utilisation de mouvements.

4- Straight Ahead and Pose to Pose (toute l'action d'un coup et partie par partie)

5- Follow Through and Overlapping Action (continuité du mouvement initial et chevauchement de deux mouvements consécutifs)

6- Slow in and slow out (ralentissement en début et en fin de mouvement)

"ease-in" et "ease-out"

7- **Arc** *La nature ne crée pas de lignes droites,*

8- Secondary Action (action secondaire)

9- Timing Les objets doivent se déplacer à des **vitesses crédibles** par rapport à leur taille et à leur masse. On contrôle le timing des objets par rapport à leurs tailles relatives, ainsi que par la durée de l'animation.

10- **Exaggeration (exagération)** Aller un peu au-delà des limites naturelles permet de donner un peu de **caractère** et de personnalité à une animation

11- Solid Drawing (notion de volume)

12- Appeal (charisme) ajouter quelques effets supplémentaires.

Chapitre 5 : Créez des transitions CSS à propriétés multiples

Il ne s'agit pas simplement de passer d'un état A à un état B. Mais davantage de faire une superposition d'effets de mouvement, afin d'obtenir une animation qui paraît bien plus authentique aux yeux de l'utilisateur.

En séparant les transitions, il devient possible d'attribuer des durées spécifiques **à chaque effet, ce qui nous permet d'ajouter de la complexité à nos animations, les rendant alors plus intéressantes et plus engageantes.**

Résumé :

- Décaler le minutage de nos animations, à la fois en modifiant leur durée et en retardant leur déclenchement, nous permet de créer des effets visuels plus authentiques, plus intéressants et plus captivants. il est possible d'**animer** autant de propriétés que l'on veut avec les transitions ;

- le mot clé `all` permet d'appliquer des transitions **simultanément** à toutes les propriétés ;
- ou bien on peut séparer les animations par **des** virgules, ce qui permet de leur donner des valeurs différentes. Exemple : `transition: transform 450ms, background-color 300ms;`
- on peut **décaler** le départ des transitions avec `transition-delay` ;
- la valeur de `transition-delay` peut également être définie directement dans la propriété `transition`.

Chapitre 6 : **Créez des animations plus naturelles avec les fonctions de timing**

Tout objet en mouvement doit forcément **accélérer**, et avant de pouvoir s'arrêter, il doit **décélérer**. Dans le jargon de l'animation, il s'agit du **ease-in** et du **ease-out**.

En CSS, on appelle les courbes d'accélération des ***fonctions de timing***.

lorsqu'aucune fonction de timing n'est précisée, le navigateur applique la fonction **ease par défaut**, `ease` comprend un `ease-in` léger puis un `ease-out` un peu plus marqué.

Les coordonnées que nous indiquons dans `cubic-bezier()` représentent la force et la direction dans laquelle nous voulons faire dévier la ligne de notre courbe d'accélération.

les effets de timing ne sont pas uniquement fonctionnels pour donner un côté naturel à nos animations, mais ils peuvent également parfois transmettre une véritable émotion et raconter une histoire.

Pour générer les coordonnées de la courbe cubic-bézier, on utilise un outil en ligne, comme le bien nommé cubic-bezier.com.

Il existe plusieurs fonctions de timing à découvrir sur MDN

Résumé :

- l'**accélération** et la **décélération** des transitions sont contrôlées par la propriété `transition-timing-function` ;
- si aucune fonction de timing n'est indiquée, la transition utilisera la fonction `ease`. Elle suit un profil d'accélération plus net, et une rampe de décélération plus prononcée ;
- parmi les autres mots clés pour les fonctions de **temporisation**, on peut trouver `ease-in`, `ease-out`, `ease-in-out`, et `linear` ;
- quand aucune fonction de timing par défaut ne correspond à l'animation, il est possible de créer ses propres courbes d'animation personnalisée à l'aide de la fonction `cubic-bezier()` ;
- il existe des outils pour ajuster les effets de timing avec la fonction `cubic-bezier()` .

Chapitre 7 : **Optimisez les performances de votre navigateur pour vos animations CSS**

Les images par seconde, ou *FPS*, représentent le nombre d'images individuelles affichées en une seconde. Plus les FPS sont élevées, plus le mouvement paraît fluide. Comme nous allons bientôt le voir, en animation web, on vise 60 FPS, mais en raison des limitations techniques de GIF, nous utilisons des exemples à 50 FPS.

- Résumé :
- à l'écran, il n'y a pas de véritable mouvement, mais une **succession d'images** s'enchaînant suffisamment rapidement pour être interprétées par notre cerveau comme du mouvement ;
- cette succession d'images s'appelle les FPS (Frame Per Second, ce qui signifie *images par seconde*) ;
- plus le FPS est **élevé**, plus l'animation est **fluide** ;
- le taux d'images par seconde idéal est **60 FPS** ;
- les quatre étapes de la création d'une page web sont :
    - **Style** : le navigateur comprend la structure HTML du code qu'il reçoit et prépare le style qui sera appliqué,
    - **Layout** (mise en page) : le navigateur détermine la mise en page et la taille des éléments en fonction du style qu'il a reçu,
    - **Paint** : il transforme les éléments en pixels,
    - **Composition :** il combine tous les éléments pour composer la page qui s'affiche ;
- pour assurer la **fluidité** des animations, il faut se contenter d'animer des propriétés de l'étape composition. Les plus utiles sont `transform` et `opacity` .

Chapitre 8 : **Créez des animations fluides avec la propriété CSS transform**

1/ Transform : Scale ( )

Donner une seule valeur à la fonction `scale()` modifie la **taille** de l'élément de manière **uniforme**, aussi bien en hauteur qu'en largeur. Mais on peut également lui préciser deux valeurs, une pour modifier la largeur (X), et une autre pour modifier la hauteur (Y).

Quand on veut modifier **l'échelle** dans une seule direction, on peut utiliser les fonctions `scaleX()` et `scaleY()`.

Exemple : ***transform: scale(1);***

2/ Transform : translate ( )

`translate()` prend deux valeurs en paramètres. La première indique la distance à laquelle on veut se déplacer sur l'axe X, et la seconde la distance sur l'axe Y. Quand on

utilise des pourcentages avec `translate()`, ces pourcentages sont liés à l'élément lui-même.

Exemple : transform: translateY(0);

3/ Transform : Rotate ( )

On peut paramétrer `rotate()` en plusieurs unités, les degrés exprimés en "deg" et les turns...

Exemple : transform: rotate(-1turn);

On ne peut assigner qu'une seule propriété `transform` à un élément.

Combinez les fonctions scale, position et :rotate !

Pour créer un effet qui combine rotation et changement d'échelle, nous devons indiquer la fonction `rotate()` à la suite de `scale()` dans la propriété `transform`

Exemple : transform: scale(.1) rotate(-90deg);

L'utilisation de plusieurs fonctions a un bémol : l'ordre dans lequel on assigne les fonctions peut avoir un gros impact sur le résultat final. Il faut savoir que le navigateur effectue les calculs de chaque fonction dans l'ordre, **de droite à gauche**.

4/ transform :skew( ) incliner des objets horizontalement ou verticalement. Penche les bords horizontaux ou verticaux, ou même les deux, en utilisant les fonctions `skewX()`, `skewY()`, et `skew()`

Exemple : **`transform: skewX(45deg); transform: skew(45deg, 45deg);`**

Passez dans une nouvelle dimension

Pour effectuer une transformation d'échelle en 3D sur l'axe Z, il faut d'abord l'avancer vers l'avant ou l'arrière en utilisant `translateZ()` ou `translate3d()`.
Sinon, `scaleZ()` ou `scale3d()` verrait son échelle modifiée de 0... ce qui n'aurait aucun effet, évidemment.

`Transform : perspective()`

La valeur qu'on donne à `perspective()` indique au navigateur à quelle "distance" se trouve le spectateur. Comme dans le monde réel, plus un objet est proche, plus le mouvement aura l'air important, alors qu'à l'inverse, un objet distant semblera plus statique.

Exemples : **transform: perspective(75px) rotateX(45deg);**

Résumé :

- la propriété `transform` nous permet de manipuler et animer nos sites de presque toutes les manières, et comme tout se passe pendant l'étape composition, les animations sont bien fluides sur tous les supports ;

- on peut déplacer des éléments avec les fonctions translate : `translate()`, `translateX()`, `translateY()` et `translate3d()` ;
- on peut agrandir avec les fonctions scale : `scale()`, `scaleX()`, `scaleY()` et `scale3d()` ;
- et on peut les faire pivoter grâce aux fonctions rotate : `rotate()`, `rotateX()`, `rotateY()` et `rotateZ()` ;
- si on ajoute une deuxième propriété `transform`, elle annule la première. On ne peut donc définir qu'une seule propriété `transform` dans un même sélecteur ;
- pour effectuer plusieurs transformations, on peut les lister dans une même propriété transform comme

```
transform:translateX(200%) scale(2)
```

;

- une propriété avec plusieurs fonctions exécute les fonctions dans l'ordre, de droite à gauche ;
- les fonctions de transformations en 3D comme `translate3d()`, `rotateZ()` et `scale3d()` ont également besoin de la fonction perspective pour indiquer au navigateur la distance à laquelle l'utilisateur se trouve : plus la distance est grande, moins l'animation sera marquée.

Chapitre 09 : **Modifiez le point d'ancrage d'un élément grâce à transform-origin**

Découvrez le point d'ancrage

Par défaut, toutes les fonctions de `transform` partent du **centre** de l'élément.

`transform-origin` permet de déplacer un **point d'ancrage** où on veut, pour faire partir nos animations de ce point. On assigne deux valeurs, une pour X et l'autre pour Y. Une seule valeur peut aussi suffire. Si cette valeur est un nombre, elle s'appliquera à l'axe X et laissera Y à la valeur par défaut de 50 %.

Les valeurs de `transform-origin` ne se limitent pas à des longueurs ou des pourcentages.
On peut aussi utiliser des mots clés CSS pour définir les points d'ancrage, comme `left` pour le mettre sur le bord gauche, ou `right` pour le mettre à droite `top` et `bottom`.
Exemples ***:*** ***transform-origin : 0% top ; (bord haut à gauche)***
transform-origin : 50% top ; (bord haut au milieu)
transform-origin : 100% top ; (bord haut à droite)
transform-origin : 0% top ; (bord haut à gauche)
transform-origin : 10px 100px ; (point d'ancrage à 10px à doite et 100px en bas du point d'origine qui est le centre)

Passez dans la troisième dimension

l'axe Z DOIT obligatoirement être défini avec des unités réelles comme les pixels, les centimètres, etc.

exemple : **transform-origin: center bottom 7.5vw;**

Résumé :

- `transform-origin` permet de **repositionner** le point d'ancrage, qui se trouve par défaut au centre de l'élément ;
- on peut **régler** ce point d'origine en utilisant des unités comme px, rem, vh, etc. ;
- il est aussi possible d'utiliser des pourcentages pour X et Y ;
- ou encore, on peut utiliser des mots clés : `left` et `right` pour l'axe X, `top` et `bottom` pour l'axe Y, et `center` pour les deux ;
- il est possible de ne pas indiquer la valeur de l'axe Y ou, quand on utilise des mots clés, de mettre uniquement une valeur : le navigateur comprend de lui-même à quel axe la valeur s'applique ;
- quand on change le point d'origine en 3D, la valeur de Z doit être exprimée en unités (et non en pourcentages) !

Analysez la performance de vos animations avec Chrome DevTools

Les animations sont probablement parfaitement fluides sur une machine puissante équipée d'une carte graphique moderne … Mais comment savoir si l'animation sera bien fluide sur la tablette bas de gamme… Chrome propose toute une suite d'outils intégrés DevTools entre autres : l'outil de performance de **Chrome DevTools** pour que nos animations soient aussi fluides que possible.

Au-delà de l'onglet **Elements**, DevTools comporte toute une série d'autres onglets pour inspecter, déboguer et analyser divers aspects de nos sites. Ici, c'est l'onglet **Performance** qui va nous intéresser. Il nous permet d'enregistrer et d'analyser comment une page se charge, réagit, et s'anime.

La propriété transform nous permet de créer des animations abouties en évitant des calculs complexes pour notre navigateur. Et moins de calculs signifie plus d'images par seconde, donc une meilleure fluidité ! 🚀

Résumé :

- **Chrome DevTools** est l'outil de prédilection des développeurs. Il permet d'**inspecter** le code source d'une page, d'**analyser** les performances de notre page, de **brider** la performance de notre machine pour simuler un appareil plus lent. Pour cette dernière option, activez l'option "CPU throttling" ;
- l'outil **Performance** permet d'analyser les performances d'une page, notamment le taux d'images par seconde d'une animation ;
- on peut utiliser l'onglet Performance pour analyser nos animations, ce qui permet de repérer les problèmes dans notre code qui pourraient causer des problèmes de fluidité sur certains supports ;

- **zoomer** sur une image précise d'une animation permet de détailler les calculs effectués par le navigateur, que nous avions vus dans le chapitre sur le fonctionnement du navigateur.

Chapitre 09 : **Animez les couleurs de manière performante avec opacity**

il faudra adopter une approche un peu différente dans la manière d'écrire notre code.

Jusqu'ici, nous avons écrit notre HTML structurellement, c'est-à-dire que chaque élément fait partie intégrante de notre layout.

Ici, nous allons devoir mettre la fonctionnalité en avant, en créant des éléments non pas pour le layout, mais uniquement pour servir notre animation.

Les changements de couleur sont une composante essentielle de l'expérience utilisateur sur un site.

Mais plutôt que de faire changer la couleur instantanément, l'ajout d'une courte transition pourrait rendre l'état `:hover` un peu plus fluide et naturel.

L'animation de la propriété `background-color` déclenche un nouveau calcul de paint à chaque image de la transition. si notre animation était plus complexe, nous aurions sans aucun doute perdu nos 60 FPS. `background-color` n'est donc pas la meilleure option pour animer des changements de couleur. La propriété à utiliser, c'est `opacity`.

Opacity : permet de modifier la transparence d'un élément et de ses enfants sur une échelle de 0 à 1, la valeur 0 représentant la transparence totale, et la valeur 1 une opacité complète.

La solution est de structurer notre bouton de manière à créer deux arrière-plans empilés l'un sur l'autre. La couche du fond serait de la couleur normale, inactive, et la couche du dessus serait de la couleur plus sombre pour `:hover`. On pourrait ensuite faire apparaître et disparaître la couche du haut en utilisant la propriété `opacity`, en créant une animation entre les deux couleurs.

L'effet est le même qu'en animant la couleur d'arrière-plan, tout en étant bien plus rapide à calculer ! Eh oui, aucun travail de paint ! La propriété opacity qui assure une performance optimale de notre navigateur.

Exploitez la puissance du CSS

::first-letter : sélectionne la première lettre de la première ligne d'un bloc, si elle n'est pas précédée par un quelconque autre contenu (comme une image ou un tableau en ligne) sur sa ligne. Le pseudo-élément `::first-letter` n'a un effet que sur les éléments ayant une valeur `display` correspondant à `block`,

P ::first-letter { color :red ; font-size :15px ; }

`::first-line :` applique la décoration à la première ligne d'un élément. La quantité de texte sur la première ligne dépend de nombreux facteurs, comme la largeur des éléments ou du document, mais aussi de la taille du texte. Comme tous les pseudo-éléments, les sélecteurs contenant `::first-line` ne ciblent pas un élément HTML réel.

`::selection :` permet d'appliquer des règles CSS à une portion du document qui a été sélectionnée par l'utilisateur (via la souris ou un autre dispositif de pointage (double clic)).

Le pseudoélément `::after` et son frère `::after` *Ajoutent du contenu HTML à la fin ou au début d'un élément HTML*

En CSS2, les pseudoéléments étaient créés avec un signe deux-points unique, comme les pseudosélecteurs, ce qui n'était pas très clair. Pour aider à les distinguer, CSS3 a modifié la syntaxe des pseudoéléments avec le préfixe double deux-points.

L'ajout des éléments `::before` ou `::after` crée un élément enfant à chaque fois que son sélecteur a été assigné. L'élément créé par `::before` sera le premier enfant de l'élément, et celui créé par `::after` sera le dernier.

Maîtrisez la particularité des pseudoéléments

les pseudoéléments nécessitent une propriété dont les éléments normaux n'ont pas besoin. Mais l**es pseudoéléments** ont besoin de la propriété **`content`** pour fonctionner. Elle permet de remplir un pseudoélément de texte ou d'images, ou pas…

Découvrez les dégradés

On peut aussi animer des dégradés au lieu de couleurs unies grâce à Opacity. Pour cela, il faut créer un dégradé pour la `background-color` du pseudoélément `::after`, et non une couleur unie.

La morale de cette histoire, c'est que la propriété `opacity` permet de faire des transitions de couleur sans que le navigateur fasse des calculs de type "paint". La performance reste donc optimale, et nous permet donc de garder notre objectif de 60 FPS. Pour cela, les pseudoéléments nous évitent d'écrire un code HTML répétitif, dans lequel on aurait dû insérer de véritables éléments supplémentaires.

Résumé :

- animer la couleur d'une propriété déclenche des **calculs** de paint ;
- la propriété `opacity` nous permet de faire des transitions entre des couleurs en évitant ces calculs ;
- la propriété `opacity` reçoit une valeur entre 0 et 1, 0 étant complètement transparent et 1 complètement opaque ;
- pour éviter d'avoir à ajouter des `<div>` supplémentaires, que l'on aurait dû ajouter à chaque fois dans le HTML, on peut utiliser le pseudoélément `::before` ou `::after` ;
- pour créer un pseudoélément, ajoutez le nom du pseudoélément à un sélecteur, en utilisant le préfixe double deux-points : `.selector::after{...}`

- les pseudo-éléments `::before` et `::after` créent un élément qui est respectivement le premier ou le dernier enfant de l'élément sélectionné ;
- il est possible de créer des dégradés de couleur. Il suffit d'attribuer un dégradé au background-color de l'élément d'arrière-plan. On fera ensuite disparaître l'élément superposé avec opacity: 0.

Chapitre 10 : Créez des animations plus complexes avec la règle CSS @keyframes

Créez des étapes supplémentaires

Alors que les transitions CSS ne font que passer d'une valeur à l'autre, les @keyframes nous permettent de concevoir des animations avec plusieurs étapes, et ainsi de créer des animations plus **complexes** et **dynamiques**.

Contrairement aux transitions, qui sont à usage unique et qui n'existent qu'à l'intérieur du sélecteur où elles ont été déclarées, les @keyframes sont disponibles globalement, donc n'importe quel sélecteur dans notre fichier CSS peut les utiliser.

Nous déclarons les @keyframes au niveau de base du fichier CSS, en utilisant l'opérateur @keyframe, suivi du nom de notre choix, entre accolades ouverte et fermée : @keyframes progress-bar { }

Définissez vos keyframes

Un keyframe CSS est défini par le **pourcentage d'animation** complété lorsque sa valeur est réalisée.

Le début de notre animation en keyframes aurait une progression de 0 % et la propriété transformée une valeur de `scaleX(0)`. La fin aurait une progression de 100 % et une valeur de `scaleX(1)`

Contrairement aux transitions, qui sont à usage unique et qui n'existent qu'à l'intérieur du sélecteur où elles ont été déclarées, les @keyframes sont disponibles globalement, donc n'importe quel sélecteur dans notre fichier CSS peut les utiliser.

Et comme ils sont disponibles de manière globale, ils ne sont pas déclarés dans un sélecteur. Au lieu de cela, nous déclarons les @keyframes au niveau de base du fichier CSS, en utilisant l'opérateur @keyframe, suivi du nom de notre choix, entre accolades ouverte et fermée :

```
@keyframes progress-bar {


 }
```

Lorsque nous utilisons la règle @keyframes, nous déclarons un **ensemble** de keyframes et nous lui donnons un nom, que nous pouvons utiliser pour appeler l'animation sur le sélecteur de notre choix.

Chaque keyframe est défini en utilisant son pourcentage, puis sa propre paire d'accolades qui nous permettent de définir les propriétés que nous souhaitons appliquer à ce stade de l'animation :

```
@keyframes progress-bar{

    0% {

        transform: scaleX(0);

    }

    100% {

        transform: scaleX(1);

    }

}
```

L'animation « progress-bar » définie par la règle keyframes part d'une échelle `scaleX()` de zéro qui finit à 1 au cours de son animation.

Dans le cas où nous voulions des keyframes de début et de fin sans rien au milieu, nous pouvons définir les keyframes en utilisant les mots clés « **from** » et « **to** » et pas besoin de pourcentage.

Généralement, on pourra utiliser la propriété raccourcie `animation` pour définir l'ensemble des propriétés liées aux animations.

`animation-name` : définit une liste d'animations qui doivent être appliquées à l'élément ciblé. Chaque nom indique une règle @ `@keyframes` qui définit les valeurs des propriétés pour la séquence.

`animation-duration` : définit la durée d'une animation pour parcourir un cycle. Elle ressemble beaucoup à `transition-duration`, et accepte une valeur en ms (millisecondes) ou en s (secondes).

Pour bien comprendre :

les transitions s'animent lorsqu'on assigne une valeur à une propriété et que cette valeur change dans l'élément déclencheur. Pour les transitions, les valeurs doivent donc être assignées aux sélecteurs à l'intérieur de notre code.

Les animations @keyframes sont un peu différentes. Lorsque `animation-name` et `animation-duration` sont assignés à un sélecteur, les propriétés et valeurs contenues dans chaque keyframe sont appliquées pendant toute la durée de l'animation.

Résumé :

- les animations @keyframes nous permettent de construire des animations plus complexes en créant plusieurs étapes pour les propriétés tout au long de l'animation ;
- les keyframes CSS sont instanciées à l'aide de la règle @keyframes suivie d'un nom pour l'ensemble des keyframes :

    - @keyframes example-frames {...} ;
- chaque keyframe peut être établi en utilisant comme valeur le pourcentage d'animation réalisé :
    - 33% {...} ;
- si vous n'avez besoin que d'un keyframe de début et de fin, les mots clés « from » et « to » peuvent être utilisés à la place de 0 % et 100 % respectivement ;
- si aucun keyframe de début ou de fin n'est fourni, l'animation commence et/ou se termine avec les valeurs de propriété assignées au sélecteur. Si aucune valeur n'est explicitement assignée dans le sélecteur, c'est la valeur par défaut qui est choisie ;
- une animation définie par la règle @keyframes peut contenir différents keyframes avec des propriétés distinctes ;
- plusieurs pourcentages peuvent être assignés à un seul keyframe. Les valeurs définies dans ce keyframe seront appliquées à ces pourcentages dans l'animation ;
- les propriétés et valeurs d'un ensemble de keyframes remplaceront les valeurs de propriétés attribuées à un sélecteur au cours de l'animation.

Chapitre 11 : Utilisez les propriétés de l'animation CSS

Comprenez la différence entre @keyframes et transition

transitions et animations sont deux techniques ayant des logiques différentes, car elle ne répondent pas au même objectif**.** Lorsque nous créons une transition, nous disons au navigateur d'aller et venir entre les valeurs assignées dans un sélecteur et les valeurs assignées à une pseudoclasse. Les @keyframes, quant à eux, lisent une série de valeurs **définies**. Une fois déclenchée, une animation avec keyframes progresse jusqu'à ce qu'elle soit terminée, ou jusqu'à ce qu'elle soit interrompue.

Les transitions sont **intrinsèquement** liées à l'état d'un élément et aux différences dans ses valeurs assignées. Les propriétés et valeurs des keyframes sont quant à elles codées en dur, elles ne nécessitent qu'un événement pour les déclencher.

Déclenchez vos animations dès le chargement de la page

Lorsque nous assignons une animation avec keyframe à un élément, il déclenchera l'animation au chargement dans le navigateur.

La propriété `animation-delay` fonctionne comme `transition-delay`, sauf qu'elle retarde uniquement les animations conçues avec des @keyframes.

Des sites comme **unsplash.com** ou **pexels.com** mettent à votre disposition des tonnes d'images de qualité, libres de droit et donc disponibles gratuitement ! Assurez-vous simplement que la licence sur la photo corresponde à l'utilisation que vous comptez en faire sur votre site.

Enchaînez les animations

Étendez les propriétés de vos keyframes avec animation-fill-mode

Résumé :

- les animations CSS @keyframes peuvent être déclenchées en utilisant des pseudoclasses telles que `:hover`, tout comme les transitions ;
- les @keyframes CSS peuvent également être déclenchés par le chargement des éléments auxquels ils sont assignés, comme un sélecteur. Par exemple, dès le chargement d'une page ;
- nous pouvons retarder le démarrage des animations avec keyframes en utilisant la propriété `animation-delay`, avec un délai exprimé en secondes ou en millisecondes, tout comme les transitions ;
- nous pouvons étendre ces valeurs du début à la fin de ces animations en utilisant la propriété `animation-fill-mode` :
    - le mot clé « backwards » prolonge les valeurs de départ d'une animation avant son lancement, couvrant la durée du délai assigné avant que l'animation elle-même ne commence,
    - le mot clé « forwards » prolonge les valeurs finales d'une animation jusqu'à ce que la page soit rechargée ou que le navigateur soit fermé,
    - le mot clé « both » prolonge l'animation dans les deux sens ;
- nous pouvons définir une fonction de timing des @keyframes en utilisant la fonction animation-timing-function sur le sélecteur où l'animation a été assignée ;
- nous pouvons également définir un timing spécifique keyframe par keyframe, en assignant la propriété `animation-timing-function` aux keyframes en question.

Chapitre 12 : Manipulez et réutilisez les animations CSS

Nous *pourrions* bien sûr créer une animation avec ***BEAUCOUP*** de keyframes...

Créez des émotions avec vos animations

Il y a plusieurs décennies, le professeur de psychologie Albert Mehrabian publiait le livre *Silent Messages,* dans lequel il détaillait les composants d'une communication efficace, en établissant la règle des 7 % - 38 % - 55 %. Selon cette règle, une communication efficace passe à 7 % par les mots, à 38 % par le ton, et à 55 % par le langage corporel. En d'autres termes, le composant le plus important est le langage corporel, mais les sites Internet nous limitent au langage. Avec l'animation, nous pouvons donner aux sites Internet la possibilité de parler avec les mains !

Créez votre premier loader

Faites boucler vos animations à l'infini

Propriété	Définition	Valeurs	Exemples
transform	Passer d'un état A à un état B	**Scale :** (échelle) Echelle par défaut=1	Transform :scale(1.5)

transition	Créer une transition	La syntaxe de la propriété raccourcie `transition` est :
		`div {`
		`transition: <property> <duration> <timing-function> <delay>;`
		`}`	

Transition-proprety	Désigner la propriété sur laquelle sera appliquée la transition		transition-property: transform;
Transition-duration	indiquer la durée de la transition.	S : secondes, ms :millisecondes	transition-duration: 1s;

p:hover + .ball	La transition sera appliquée à l'élément .ball voisin direct de p lorsque ce dernier est survolé		P :hover + .ball {
			Background-color :red ;
			}

All
déclencher plusieurs changements au même moment et pour la même durée
	Mot clé qui indiquer au navigateur d'appliquer la transition à toutes les propriétés que nous avons modifiées au sein de la pseudoclasse `:hover`		P {transition: **all** 450ms;}
			:hover {
			transform: scale(1.13);
			background-color: red;
			}

`transition-delay`	**retarde** le début de l'animation de transition par la valeur de temps que vous indiquez		transition-delay: 150ms;
`transition-timing-function`	**Fonction de timing**	**Linear : linéaire** se déplace à **vitesse constante** du départ à l'arrivée **`ease-in-out`** courbe d'**accélération** et de **décélération subtile**, `ease-in` accentuer l'accélération au début `ease-out`.	transition : transform 1000ms linear ; transition : transform 1000ms ease-in-out ; transition : transform 1000ms ease-in ; transition : transform 1000ms ease-out ;

		accentuer la décélération à la fin	

cubic-bezier ()	**fonctions de timing**
	a besoin d'une liste de quatre nombres, qui sont les coordonnées de deux points sur le graphique. Les deux premiers nombres sont les **coordonnées** X et Y du point qui déterminent le `ease-in` de la courbe d'accélération, et les deux suivants déterminent le `ease-out`
			.selector {
			transition-timing-function : **cubic-bezier(.42, 0, .58, 1) ;**
			}

Animation	**Animation-name, animation-duration,** animation-timing-function, animation-delay, animation-iteration-count, animation-direction, animation-fill-mode, animation-play-state.		
Animation-name	définit une liste d'animations qui doivent être appliquées à l'élément ciblé. Chaque nom indique une règle @ `@keyframes` qui définit les valeurs des propriétés pour la séquence.		Animation-name : couleur ;
Animation-duration	définit la durée d'une animation pour parcourir un cycle		**Animation-duration : 200ms ;**
animation-timing-function	définit la façon dont une animation CSS doit se dérouler au fur et à mesure de chaque cycle.	Linear ; ease-in-out ; step(5,end) ; cubic-bezier(0.1,-0.6,0.2,0) ….	animation-timing-function : ease ;
animation-delay	définit la durée d'attente avant de démarrer une animation une fois qu'elle a été appliquée à un élément.		animation-delay :2s ;
animation-iteration-count	indique le nombre de cycles utilisés pour répéter une animation avant que celle-ci s'arrête.	**`Infinite :`** répétée à l'infini **`<number>`** nombre de répétitions pour l'animation	

animation-direction	indique si les cycles de l'animation doivent être joués dans le sens inverse et/ou de façon alternée.	`normal;`
		`reverse;`
		`alternate;`
		`alternate-reverse;`	

animation-fill-mode	indique la façon dont une animation CSS doit appliquer les styles à sa cible avant et	`none;`
		`forwards;` (dernière étape)
		`backwards;`
		première *keyframe* pertinente
		`both;`	

	après son exécution.		
animation-play-state.	définit si une animation est en cours d'exécution ou si elle est en pause.	`Paused ; running;`	

Maîtrisez les propriétés raccourcies

Propriété raccourcie	Raccourcie de :	
Ici, nous ne voulons pas retarder l'animation de transformation, mais celle de couleur d'arrière-plan ;		transition: transform 450ms, background-color 300ms; transition-delay: 0, 150ms;
Ici la propriété transform n'a pas de délai (elle commence à 0 et dure 450ms) tant disque background est retardée de 150ms puis durera 300ms ce qui arrêtera les deux propriétés au même moment.		**transition: transform 450ms, background-color 300ms 150ms;**
Transition : transform 400ms ;	Transition-proprety Transition-duration	
:not(:focus):invalid	Lorsque on est pas focus et invalid	
Séparez les propriétés	Une liste d'animations de chaque propriété à laquelle nous voulons ajouter une transition, séparées par des **virgules**.	transition: transform 450ms, background-color 450ms;