

Semaine 07 - Animation CSS (suite) et principes d'animation

Les fonctions temporelles avec la sous-propriété «**timing-function**»

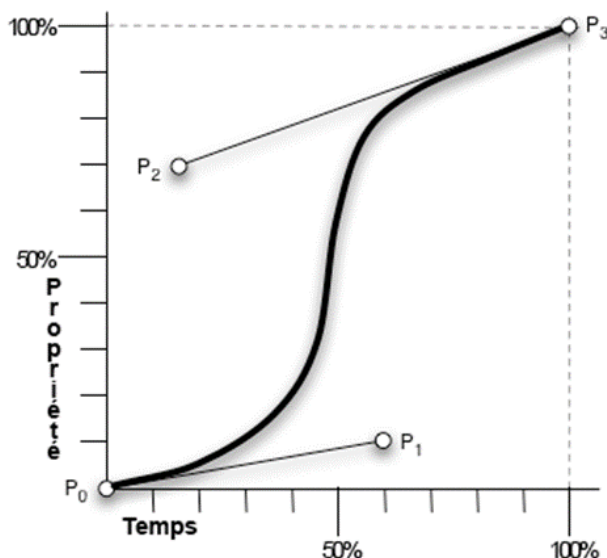
La sous-propriété « **timing-function** » définit un effet de « *timing* » à utiliser (une courbe d'accélération) entre deux propriétés CSS ou entre chaque image-clé (« *keyframe* »). La déclaration de cette propriété est facultative, mais ajoute beaucoup de réalisme à un effet d'animation ou de transition.

En effet, en animation, on se préoccupe beaucoup des changements de vitesse, c.-à-d. d'accélération et de décélération. **Pourquoi?** *Tout simplement parce qu'un personnage ou un objet ne peut pas passer de l'état arrêté à l'état en mouvement ou vice-versa sans accélérer ou ralentir...* C'est notamment la durée de l'accélération et du ralentissement qui vont rendre compte du poids d'un objet, de sa taille, et de l'énergie mise en jeu...

Ces effets sont utilisés en tant que valeur des propriétés suivantes:

- pour une **transition** CSS: **transition-timing-function**;
- pour une **animation** CSS: **animation-timing-function**.

Les effets de « *timing* » permettent de gérer la **fluidité d'une animation** entre deux propriétés CSS ou entre deux images-clés (« *keyframes* ») en contrôlant comment l'animation se déroule au cours du temps. Un effet de « *timing* » peut-être représenté sous deux formes: une **courbe de Bézier cubique** ou une **fonction en escalier** (« *stepping function* »). Dans le cas de ces deux fonctions, on pourra se les représenter sur un graphique où l'**axe des Y** représentera la **progression de la transformation** de la propriété CSS subissant une modification et, l'**axe des X** représentera l'**avancement dans le temps** de la transition (de son début à sa fin).








Source: <http://www.alsacreations.com/tuto/lire/1299-timing-des-animations-et-des-transitions-en-css3.html>

Dans le cas du graphique précédent (où une courbe de Bézier est utilisée), l'animation ou la transition commencerait lentement, accélérerait et finirait lentement.

Les fonctions temporelles prédéfinies

De base, les navigateurs fournissent quelques fonctions d'accélération par défaut. Ces fonctions pré-définies sont les suivantes :

	linear : L'animation, entre les images-clés, est stable ou constante, sans décélération ni accélération.
	ease : L'animation, entre les images-clés, décélère légèrement au début, puis accélère progressivement et termine lentement à l'approche de son état final. C'est la valeur par défaut.
	ease-in : L'animation, entre les images-clés, commence lentement, puis accélère progressivement jusqu'à son état final.
	ease-out : L'animation, entre les images-clés, démarre rapidement puis ralentit progressivement à l'approche de son état final.
	ease-in-out : L'animation, entre les images-clés, démarre lentement, accélère puis ralentit à l'approche de son état final.

Pour définir un effet de « *timing* » s'appliquant à une **transition CSS**, il faut définir la valeur de l'accélération souhaitée avec la propriété : **transition-timing-function** dans le sélecteur CSS de l'élément HTML à animer. Dans l'exemple qui suit, une accélération de type « *ease-in* » est appliquée sur la transition animant l'espacement des lettres d'un titre (balise h1) au survol de cet élément :

```
h1 {  
  font-size: 2rem;  
  letter-spacing: 0rem;  
  transition-property: letter-spacing;  
  transition-duration: 1s;  
  transition-timing-function: ease-in;  
}  
  
h1:hover {  
  letter-spacing: .3rem;  
}
```

Soulignons qu'en utilisant la propriété raccourcie pour définir la transition, on a qu'à ajouter le mot clé à la déclaration :

```
transition: letter-spacing 1s ease-in;
```

TIM 582 - 2W2 Animation et interactivité en Web

Pour définir un effet de « *timing* » s'appliquant à une **animation CSS**, il faut définir la valeur de l'accélération souhaitée avec la propriété : **animation-timing-function** dans le sélecteur CSS de l'élément HTML à animer. Dans l'exemple qui suit, une accélération de type « *linear* » est appliquée sur l'animation d'un élément HTML de type `div` traversant simplement son conteneur parent de haut en bas, et ainsi de suite :

```
/* Images-clés de l'animation */
@keyframes déplacer-carre {
  from, to {
    top: 0%;
  }

  50% {
    top: 85%;
  }
}

div {
  .....
  /*Propriétés d'animation*/
  animation-name: déplacer-carre;
  animation-duration: 4s;
  animation-iteration-count: infinite;
  animation-timing-function: linear;
}
```

Encore ici, en utilisant la propriété raccourcie pour définir l'animation , on a qu'à ajouter le mot clé à la déclaration:

```
animation: déplacerCarre 4s infinite linear;
```

Il est important de noter que dans le cas d'une **animation CSS**, si l'effet d'accélération est appliqué dans le **sélecteur CSS**, celui-ci sera appliqué pour la transition **entre chaque image-clé**, ce qui peut enlever du réalisme à l'effet d'animation souhaité.

Il serait en effet plus intéressant qu'un élément accélère en descendant et qu'il décélère en remontant, donc de **faire varier la courbe d'accélération d'une image-clé à l'autre**. Puisqu'il est effectivement souhaitable, dans certains cas, que les effets de « *timing* » puissent varier d'une image-clé à l'autre, la propriété «**animation-timing-function**» est la **SEULE PROPRIÉTÉ** d'une animation CSS à pouvoir être insérée **au sein d'une règle @keyframes**.

L'exemple qui suit, bonifie l'effet d'animation du code précédent en appliquant un effet d'accélération lorsque l'élément descend et un effet de décélération lorsqu'il remonte :

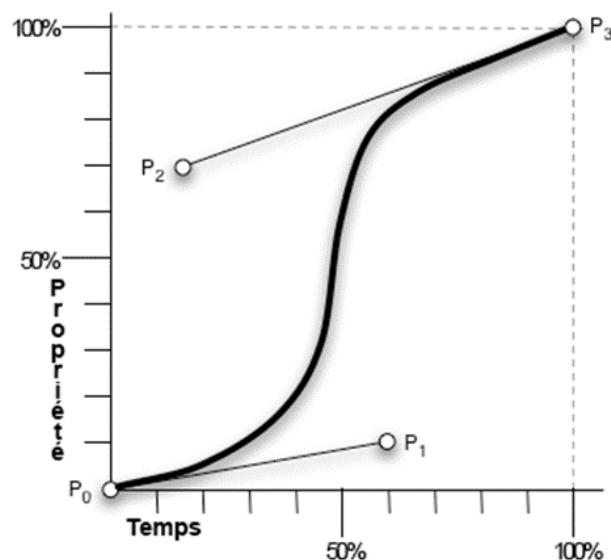
```
@keyframes déplacer-carre {
  from,
  to {
    top: 0%;
    animation-timing-function: ease-in;
  }
}
```

```
50% {  
  top: 85%;  
  animation-timing-function: ease-out;  
}  
  
div {  
  .....  
  /*Propriétés d'animation*/  
  animation: deplacer-carre 4s infinite alternate;  
}
```

Les courbes d'accélération personnalisées - la fonction CSS `cubic-bezier()`

En CSS, il est possible de définir nos propres courbes d'accélération personnalisée avec la fonction : `cubic-bezier()`.

Une courbe de **cubic-bezier** sera définie par quatre points (p_0 , p_1 , p_2 , p_3) disposés sur un graphique où les axes X (représentant le temps) et Y (représentant les variations de valeurs sur les propriétés CSS animées) auront pour valeur une échelle entre 0 et 1. Les points **p_0** et **p_3** auront toujours pour valeur de positionnement respective (0,0) et (1,1). En soit, le **p_0** représente le **début de la transition** et le **p_3** représente la **fin**. Ces deux points sont fixes et ne peuvent pas être modifiés.



Ainsi, la fonction **cubic-bezier** nous sert à **définir la position en X et en Y des points 1 et 2** qui représentent les « poignées » nous aidant à définir la forme de la courbe sur le graphique. La fonction prendra en paramètre 4

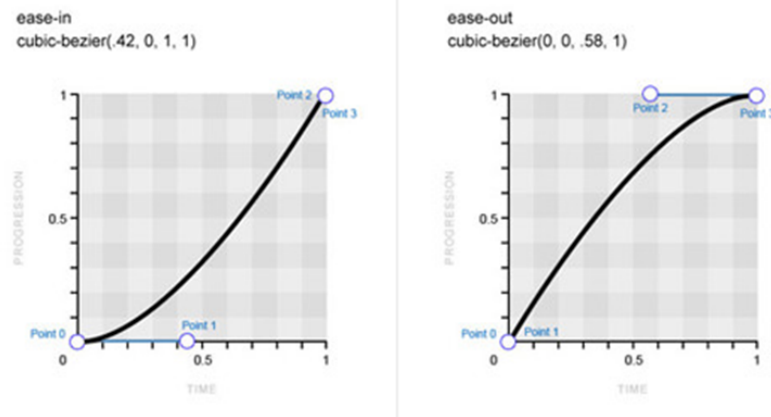
nombre flottant représentant respectivement les coordonnées X et Y du point 1 et les coordonnées X et Y du point 2:

animation-timing-function: cubic-bezier(<p1 X>, <p1 Y>, <p2 X>, <p2 Y>);

Exemple :

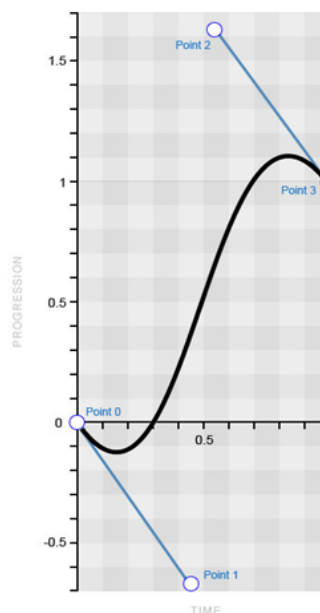
animation-timing-function: cubic-bezier(0.25, 0.1, 0.25, 1.0);

Il est à noter que les fonctions d'accélération offertes par défaut par les navigateurs correspondent également à une fonction `cubic-bezier()`. La figure qui suit illustre les valeurs numériques de deux d'entre-elles:



La **valeur de position sur l'axe X** (représentant le temps) **des deux points** doit avoir une valeur située entre 0 et 1. La **valeur en Y peut cependant prendre n'importe quelle valeur**. L'intérêt premier de pouvoir spécifier n'importe quelle valeur sur l'axe des Y est de permettre très facilement l'utilisation d'effets de rebond (« *bouncing* ») dans divers cas. Exemple d'une telle courbe :

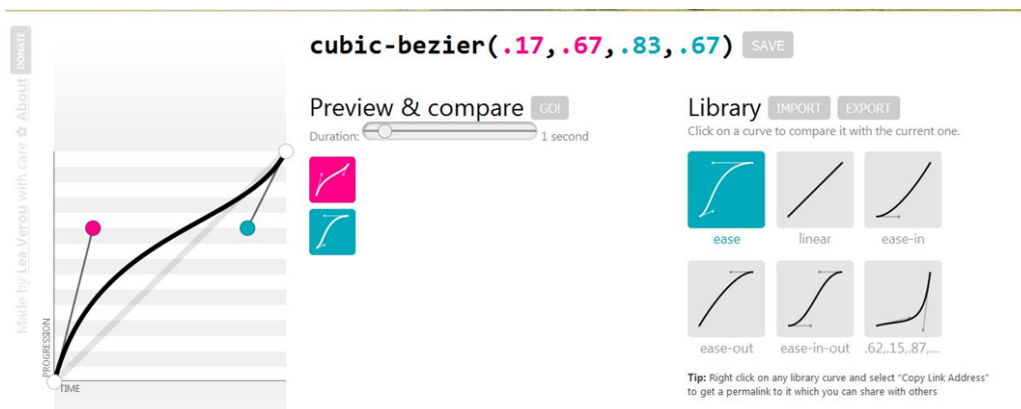
Custom Cubic Bézier Curve Outside 0-1 Range
`cubic-bezier(.45, -0.67, .53, 1.63)`



Outils pour générer ses propres courbes de Bézier

Toute cette théorie semble peut-être un peu compliquée. Il peut aussi être très difficile de se représenter visuellement la courbe d'accélération qui pourrait résulter des valeurs numériques indiquées en paramètres de la fonction `cubic-bezier()`. Mais **RASSUREZ-VOUS**, plusieurs outils en ligne vous permettent de générer vos propres courbes de Bézier dans un environnement simple et efficace.

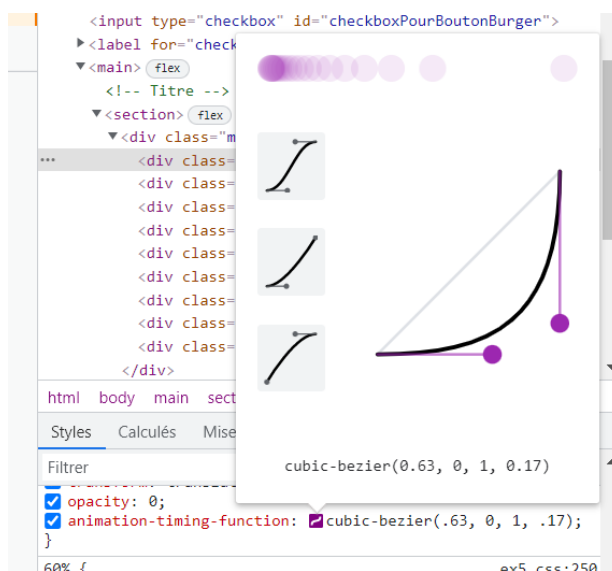
Notamment, un outil créé par Lea Verou (en anglais) accessible à l'URL suivant : <http://cubic-bezier.com> permet tout simplement de créer vos propres courbes de Bézier applicables aux effets de « timing » en animation CSS:



Cet outil présente les avantages suivants :

- Interface graphique permettant une compréhension très simple des courbes de Bézier;
- Possibilité de jouer avec la courbe jusqu'à l'obtention de l'effet désiré;
- Possibilité de tester vos courbes en direct et de les comparer;
- Possibilité d'enregistrer et de partager vos propres courbes.

À noter que les outils de développement des navigateurs permettent maintenant aussi de modifier un effet d'accélération. Nous avons uniquement à sélectionner l'élément HTML concerné et de vérifier les styles appliqués et nous aurons accès à l'outil permettant de modifier la courbe:



Les fonctions temporelles en escalier («stepping function»)

La propriété « *timing-function* » peut également définir une **transition/animation** en escalier. Cela veut dire que la transition ou l'effet d'animation ne sera pas fluide comme si elle utilisait une fonction cubic-bezier(), mais passera par des **étapes** (« **steps** ») bien tranchées.

La fonction CSS steps()

La fonction CSS **steps()** permet, tel que mentionné ci-haut, de définir une transition ou une animation en «escalier» ou **pas à pas**. Cela veut dire que la transformation entre deux propriétés CSS lors d'une transition ou d'une animation CSS sera **saccadée** et passera par des **étapes** (« **steps** ») distinctes.

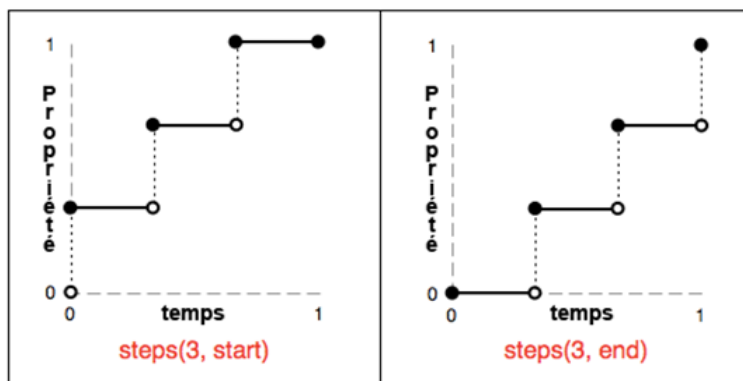
La syntaxe de cette fonction est plutôt simple :

- transition-timing-function: **steps(nombre_étapes, direction);**
- animation-timing-function: **steps(nombre_étapes, direction);**

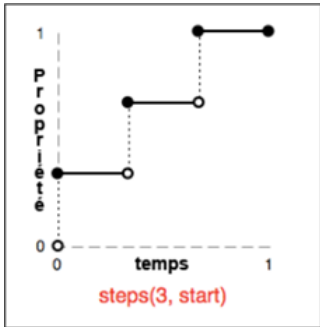
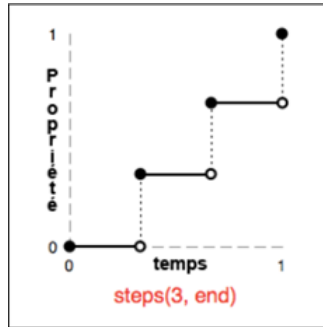










Elle prend deux paramètres :

- Un **premier paramètre** (*nombre_étapes*). Représentant **obligatoirement** un **nombre entier** définissant le nombre de « marches » ou de « pas » de la transition ;
- Un **deuxième paramètre** (*direction*): Désignant le sens de la transformation par un **mot clé** qui définira si la propriété subissant la transition sera modifiée au début ou à la fin du délai de chaque « *marche* », « *étape* » ou « *pas* ». Les principaux mots-clés sont : **start** (au début) ou **end** (à la fin). Une valeur de départ (« **start** ») modifiera la valeur de la propriété CSS au début de chaque étape, tandis qu'une valeur de fin (« **end** ») modifiera la valeur de la même propriété CSS à la fin de chaque étape. La valeur par défaut est : **end**.

Ainsi, le graphique représentant les fonctions steps(3,start) et steps(3,end) serait le suivant:



Les images qui suivent illustrent l'élargissement d'une division, en 3 étapes, dans son conteneur parent en utilisant les deux types de direction : **start** et **end** :

Temps écoulé	animation-timing-function:steps(3,start);	animation-timing-function:steps(3,end);
Paliers d'animation		
État initial :		
Temps : 0sec.		
Temps : 1sec.		
Temps : 2sec.		
Temps : 3sec.		

Il est important de noter qu'avec le mode de direction « **end** » **tous les changements d'état seront perceptibles pour une transition. Ce sera toutefois différent pour une animation.** En effet, si l'animation ne joue qu'une seule fois, la dernière étape n'est pas réellement dessinée à l'écran, sauf si le « fill-mode » de l'animation est réglé sur « *forwards* ». Dans le même ordre d'idée, si l'animation joue en boucle (« *animation-timing-function : infinite ;* »), la dernière étape de l'animation n'est jamais perceptible.

Animation d'une feuille de sprites avec CSS

Avec ou sans CSS, la technique d'animation est toujours la même : nous allons utiliser le principe du dessin animé, en faisant défiler des images fixes, ce qui donnera la sensation de mouvement.

Pour gérer des feuilles de sprites (« *Spritesheet* ») avec CSS, nous avons besoin d'utiliser la propriété CSS « *background* » et sa sous-propriété « *background-position* ».

Pour bien gérer des feuilles de sprites avec CSS, les deux règles suivantes doivent être respectées:

- Toutes les vignettes (« *frame*») de la feuille de sprites doivent être de taille égale;

- Toutes les images d'une même séquence animée, doivent être disposées sur une seule rangée (ou une seule colonne). Certains outils disposent, par défaut, les vignettes en une combinaison de lignes et de colonnes! Ce qui ne serait pas gérable en CSS.

Dans l'exemple qui suit, nous allons utiliser la feuille de sprites animant le vol d'un petit poulet.



Avant de commencer, nous devons connaître les trois informations suivantes :

- la **taille** (largeur et hauteur) de chaque vignette;
- la **largeur totale** de la feuille de sprites;
- le **nombre de vignettes** constituant la séquence animée.



Ainsi, dans notre exemple, la séquence animée est composée de **8 images**, la taille des vignettes est de **208 px X 176 px** et la largeur totale de la feuille de sprites est de **1664 px**.

Première étape : la mise en place des éléments de base

La première étape afin d'animer notre feuille de sprites est de créer un conteneur HTML et de fixer ses dimensions selon la taille des vignettes de la feuille de sprites.

Conteneur HTML:

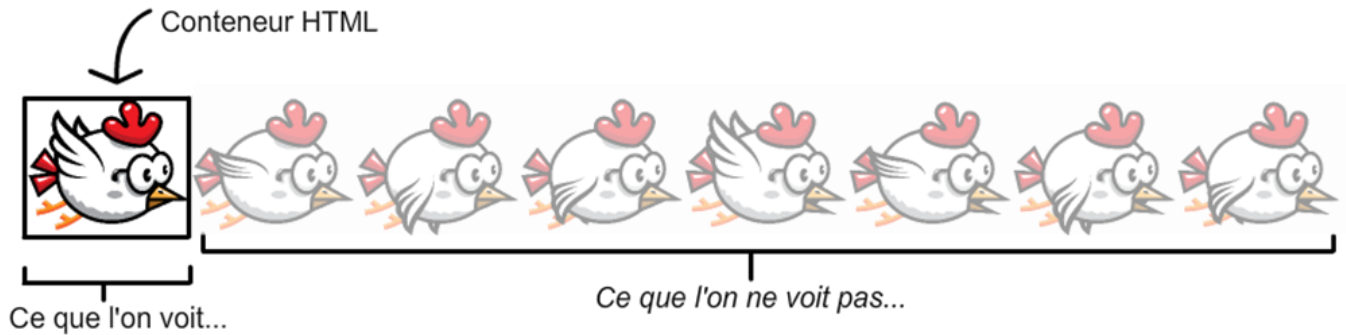
```
<div class="poulet"></div>
```

Style du conteneur HTML (dimensions et image) :

```
.poulet {  
  width: 208px;  
  height: 176px;  
  background-image: url("img/feuilleSpritesPoulet.png");  
}
```

Deuxième étape : l'animation...

Suite à l'étape précédente, il va de soit que rien ne bouge, le résultat à l'écran est le suivant :



La première image de la feuille de sprites est affichée comme une image de fond dans le conteneur HTML et, les vignettes restantes sont invisibles, car elles sont à l'extérieur du cadre. Pour l'animation, la solution est simple, il faut faire glisser l'image vers la gauche afin que le reste des vignettes s'affiche. Pour faire cela, nous allons utiliser une **animation CSS** pour animer la propriété **background-position**.

La règle **@keyframes** sera la suivante :

```
@keyframes animer-vol-poulet {  
  from {  
    background-position: 0px;  
  }  
  
  to {  
    background-position: -1664px;  
  }  
}
```

L'**image-clé 100%** définit la position X de l'image de la feuille de sprites avec une valeur négative, pour un glissement vers la gauche : **background-position : -1664px**. Cette valeur correspond à la largeur totale de la feuille de sprites. Plus spécifiquement, cette largeur doit correspondre à un **multiplicatif** du nombre de vignettes affichées. Dans notre cas, la dimension de 1664px correspond effectivement à la taille des 8 vignettes (8 X 208px = 1664px). À ce sujet, il est important de noter que, selon l'outil utilisé pour générer les feuilles de sprites, il se peut qu'on ait besoin de faire certains correctifs ou certains ajustement manuels car certains éditeurs laissent parfois des pixels résiduels (ou espace vide inutilisé par des vignettes) à la droite de l'image...

Enfin, pour animer, il suffit d'utiliser la fonction d'accélération « **steps** » avec la direction « **end** ». Le nombre d'étapes déclarées doit correspondre au nombre de vignettes constituant la séquence animée.

```
.poulet {  
  .....  
  /*Animation de la feuille de sprites du poulet*/  
  animation: animer-vol-poulet 800ms steps(8,end) infinite;  
}
```

Douze grands principes d'animation

Les artistes des studios Disney, véritables pionniers de l'animation ont, au fil de leur expérimentation, développé de nombreux concepts d'animation, dont certains sont devenus des incontournables. **Douze de ces concepts se sont imposés comme des principes fondamentaux, les piliers de l'animation.** Voici la présentation qu'en faisait John Lasseter (ancien directeur artistique de Pixar Animation Studios et de Walt Disney Animation Studios):

1. **Squash and stretch** (Écrasement et étirement). Il sert à définir la matière (rigidité et masse) d'un objet ou personnage en le déformant pendant l'action;
2. **Timing and motion** (Temps et mouvement) Placement temporel des actions, qui donne une idée de la taille et du poids d'un objet ou d'un personnage mais aussi de l'émotion véhiculée par l'action;
3. **Anticipation** - La préparation d'une action pour l'audience;
4. **Staging** (Mise en scène judicieuse) Présentation d'une idée de façon à ce qu'elle soit immanquablement claire et pour guider les yeux de l'utilisateur vers les éléments importants ;
5. **Follow through and overlapping action** (Enchaînement des actions). La fin d'une action et la mise en place de sa relation avec l'action suivante;
6. **Straight ahead action and pose to pose action** (Action directe et action de pose à pose). Les deux approches de la création du mouvement (qui correspond davantage à l'animation traditionnelle mais fait penser à la différence transitions CSS/@keyframes);
7. **Slow in and slow out** (Accélération et décélération). L'espacement des dessins d'une image à l'autre, qui permet d'obtenir de la subtilité dans le timing et le mouvement. Ce principe est basé sur le fait que les objets ne démarrent pas et ne s'arrêtent pas instantanément ;
8. **Arcs**. La forme de la trajectoire du mouvement lors d'une action naturelle;
9. **Exaggeration** (Exagération). Accentuer l'essence d'un dessin ou d'une action;
10. **Secondary action** (Action secondaire). Action résultant d'une autre action, ou secondaire du point de vue de son importance;
11. **Appeal** (Personnalité ou attrait) Créer un dessin ou une action que le public prend du plaisir à regarder;
12. **Solid drawing** (Technique de dessin développée). Apprendre à dessiner le mieux possible avant d'animer.

Ces principes sont simples et accessibles à tout le monde. Même s'ils ne garantissent pas la qualité du résultat, il vaut mieux ne pas les ignorer... même pour une animation non traditionnelle comme pour l'animation d'éléments HTML d'une page Web... Ces principes facilitent la création de **mouvements qui semblent obéir aux lois de la physique** et paraissent ainsi beaucoup **plus réalistes aux yeux des spectateurs**. À ce sujet, consulter le site Web suivant qui explique davantage chacun des principes en les appliquant à l'animation d'éléments dans une page Web :

<https://openclassrooms.com/fr/courses/5919246-creez-des-animations-css-modernes/6340915-appliquez-les-12-principes-de-l-animation-au-web>

Quelques références utiles sur les fonctions temporelles ou les principes d'animation

- Becker, Alan (Mai 2017), 12 principes of animation. Repéré à : https://www.youtube.com/playlist?list=PL-bOh8bttec4CXd2ya1NmSKpi92U_I6ZJd
- Kirupa. (mai 2015). Sprite Sheet Animations Using Only CSS. Repéré à : http://www.kirupa.com/html5/sprite_sheet_animations_using_only_css.htm
- Mozilla Corporation. (Octobre 2022). <timing-function> - CSS | MDN. Repéré à <https://developer.mozilla.org/en-US/docs/Web/CSS/timing-function>
- ONF (Mars 2015). Les 12 principes de l'animation. Repéré à : <https://blogue.onf.ca/blogue/2015/03/12/les-12-principes-lanimation/>