CSS

Transitions et animations

Transitions

🔗 Utiliser les transitions CSS

Une transition en CSS permet d'interpoler les valeurs de propriétés d'un élément depuis un état A vers un état B

Exemple de transition sur `background-color` lors d'un `:hover`

Hover me!

On utilise la propriété CSS `transition`,
dont voici la syntaxe :

```
transition: <propriété> <durée> <timing> <délai>
```

```
transition: <propriété> <durée> <timing> <délai>
```

- `<propriété>` : Nom de la propriété à animer
- `<durée>` : Durée de l'animation (`s` ou `ms`)
- `<timing>*` : Fonction de timing
- `<délai>*` : Délai avant déclanchement (`s` ou `ms`)

** (optionnel)*

Exemple d'utilisation sur un lien :

```
a {
  …
  background-color: rgba(0, 100, 200, .8);
  transition: background-color 1s;
}

a:hover {
  background-color: rgba(0, 200, 100, .8);
}
```

Mon lien

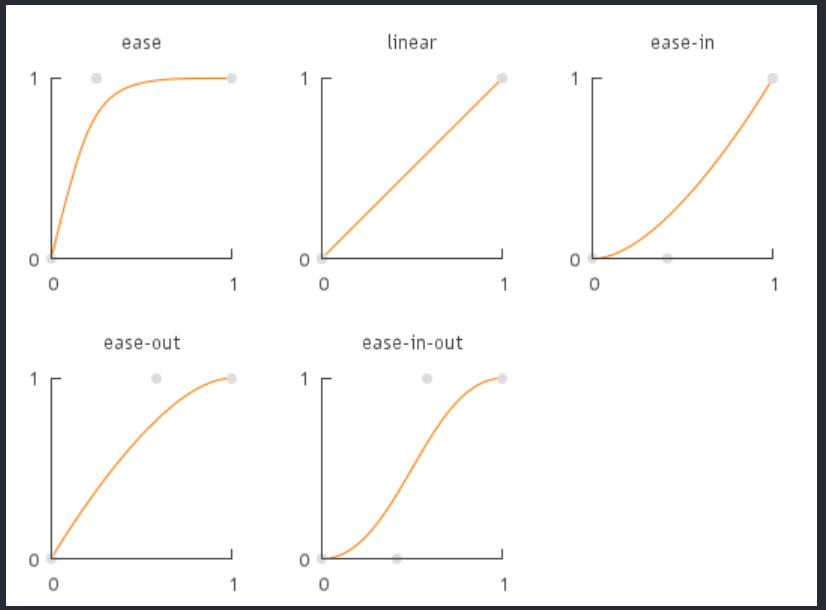
On peut utiliser `all` pour animer **toutes** les propriétés changeantes

```css
a {
  …
  background-color: rgba(0, 100, 200, .8);
  color: white;
  transition: all 1s;
}

a:hover {
  color: black;
  background-color: rgba(0, 200, 100, .8);
}
```

Mon lien

Fonctions de timing




```
transition: margin-left 2s ease;
transition: margin-left 2s linear;
transition: margin-left 2s ease-in;
transition: margin-left 2s ease-out;
transition: margin-left 2s ease-in-out;
```

Try it yourself !

Il est possible d'animer toutes les propriétés interpolables en CSS.

`margin`, `padding`, `color`, `width`, `font-size`, `box-shadow`, `border-radius`, …

Animations

🔗 Utiliser les animations CSS

Les animations permettent de faire des choses plus poussées que les transitions (comme par exemple *répéter plusieurs fois*, ou les faire passer par *plusieurs étapes*)

Pour définir une animation, il faut d'abord *définir son comportement*

Le comportement régit la façon dont va réagir l'objet animé depuis un état `0%` vers un état `100%`

Un comportement se définit à l'aide du mot-clé `@keyframes`

```
@keyframes rebond
{
  0%   { transform: translateX(0); }
  50%  { transform: translateX(900px); }
  100% { transform: translateX(0); }
}
```

L'animation nommée `rebond` déclare 3 étapes dans lesquelles on interpole la valeur de `transform`

Pour utiliser l'animation sur un élément HTML, on utilise la propriété CSS `animation`, dont voici la syntaxe :

```
animation: <nom> <durée> <timing> <délai> <répétitions> <direction>
           <fill-mode> <play-state>
```

```
animation: <nom> <durée> <timing> <délai> <répétitions> <direction>
           <fill-mode> <play-state>
```

- `<nom>` : Nom de l'animation à utiliser
- `<durée>` : Durée de l'animation (`s` ou `ms`)
- `<timing>*` : Fonction de timing
- `<délai>*` : Délai avant déclanchement (`s` ou `ms`)
- `<répétitions>*` : Nombre de répétition (nombre, ou `infinite`)
- `<direction>*` : Sens des cycles de l'animation
- `<fill-mode>*` : Etat initial et final de l'animation
- `<play-state>*` : Etat de l'animation (en pause ou en cours)

** (optionnel)*

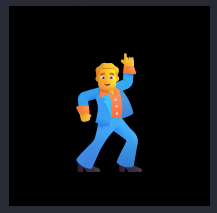
Démo 1

Démo 1

```css
@keyframes rebond
{
  0%   { transform: translateX(0); }
  50%  { transform: translateX(900px); }
  100% { transform: translateX(0) rotate(-360deg); }
}

.dancing-square {
  animation: rebond 6s linear infinite;
}
```

```html
<div class="dancing-square">🕺</div>
```



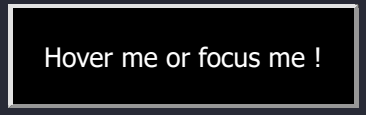
Démo 2

Démo 2

```css
@keyframes rotate {
  from { transform: rotate(0deg); }
  to   { transform: rotate(360deg); }
}

.turning-button {
  animation-name: rotate 3s linear infinite paused;
}
.turning-button:hover {
  animation-play-state: running;
}
.reveal .turning-button:focus {
  animation-direction: reverse;
}
```

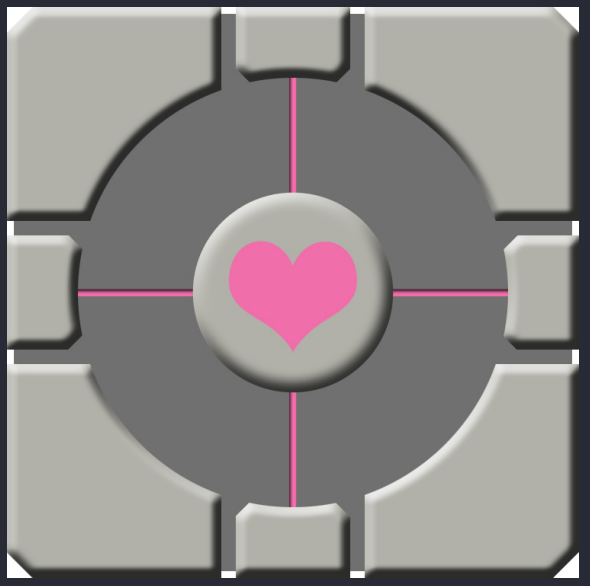
```html
<button class="turning-button">Hover me or focus me !</button>
```



Démo 3 *

** 3 comme 3D ?*

Démo 3 *



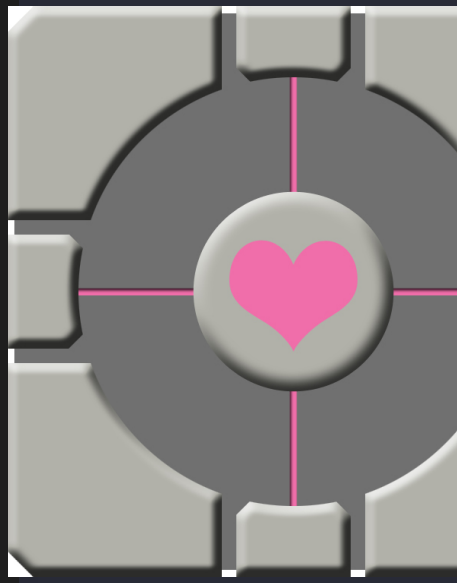
Démo 3 *

```html
<div class="scene-3d">
  <div class="cube-3d">
    <img src="cube.jpg" alt="">
    <img src="cube.jpg" alt="">
    <img src="cube.jpg" alt="">
    <img src="cube.jpg" alt="">
    <img src="cube.jpg" alt="">
    <img src="cube.jpg" alt="">
  </div>
</div>

<style>
  .scene-3d {
    --cube-size: 300px;
    --distance: 150px;

    perspective: 400px;
    width: var(--cube-size);
    height: var(--cube-size);
    margin-left: auto;
    margin-right: auto;
  }
  .scene-3d:hover .cube-3d {
    --distance: 220px;
  }

  .cube-3d {
    position: relative;
    width: var(--cube-size);
    height: var(--cube-size);
    transform-origin: 50%;
    animation: rotate-cube 6s linear infinite;
    transform-style: preserve-3d;
  }
  .cube-3d > img {
    position: absolute;
    top: 0;
    left: 0;
```



```
    width: var(--cube-size);
    height: var(--cube-size);
    transition: all 1s;
  }
  .cube-3d > img:nth-child(1) {transform: translateX(var(--distance)) rotateY(90deg); }
  .cube-3d > img:nth-child(2) {transform: translateX(calc(var(--distance) * -1)) rotateY(90deg); }
  .cube-3d > img:nth-child(3) {transform: translateY(var(--distance)) rotateX(90deg); }
  .cube-3d > img:nth-child(4) {transform: translateY(calc(var(--distance) * -1)) rotateX(90deg); }
  .cube-3d > img:nth-child(5) {transform: translateZ(var(--distance)); }
  .cube-3d > img:nth-child(6) {transform: translateZ(calc(var(--distance) * -1)); }

  @keyframes rotate-cube {
    from { transform: translateZ(-300px) rotateX(0) rotateY(0); }
    to { transform: translateZ(-300px) rotateX(360deg) rotateY(360deg); }
  }
</style>
```