Contrôler les animations avec JavaScript

Dans cette leçon, nous allons voir comment contrôler les animations avec JavaScript. Nous allons voir comment démarrer, arrêter, mettre en pause et reprendre une animation. Nous allons également voir comment gérer les événements liés aux animations. Finalement, nous allons voir comment décaler l'exécution d'une fonction.

Temporisation de fonctions

setTimeout

Pour exécuter une fonction après un certain délai, vous pouvez utiliser la fonction `setTimeout`. Cette fonction appartient à l'objet `window` et prend deux arguments : la fonction à exécuter et le délai en millisecondes. Cette fonction n'est exécutée qu'une seule fois.

```
let elementHTML = document.body;
elementHTML.classList.add("invisible");

function afficher() {
    elementHTML.classList.remove("invisible");
}

setTimeout(afficher, 1000); // Affiche "Hello !" après 1 seconde
```

setInterval

Pour exécuter une fonction à intervalles réguliers, vous pouvez utiliser la fonction `setInterval`. Cette fonction appartient également à l'objet `window` et prend deux arguments : la fonction à exécuter et l'intervalle en millisecondes.

```
let elementHTML = document.body;

function clignoter() {
    elementHTML.classList.toggle("invisible");
}

let intervalle = setInterval(clignoter, 1000); // Fait clignoter l'élément toutes
les secondes
// Il est important de l'enregistrer dans une variable pour pouvoir l'arrêter plus
tard
```

clearInterval

Pour arrêter l'exécution d'une fonction à intervalles réguliers, vous pouvez utiliser la fonction `clearInterval`. Cette fonction prend en argument l'identifiant de l'intervalle retourné par `setInterval`.

```
let elementHTML = document.body;
let compteur = 0;
function clignoter() {
    compteur++;
    if (compteur === 5) {
        clearInterval(intervalle); // Arrête le clignotement après 5 fois
    }
    elementHTML.classList.toggle("invisible");
}

let intervalle = setInterval(clignoter, 1000); // Exécute la fonction toutes les
secondes
```

Rappel des animations et transitions CSS

Les transitions CSS

Les transitions CSS permettent de modifier progressivement les propriétés CSS d'un élément. Pour définir une transition, vous devez spécifier la propriété à animer, la durée de l'animation et l'effet de transition. La transition est utile pour les changements d'état simples à deux états. Ex: une couleur de rouge à vert, une opacité de 0 à 1, etc.

On place la transition sur l'élément à animer. Lorsque la propriété change, la transition est appliquée automatiquement.

On indique la propriété à animer, la durée de l'animation et l'effet de transition. Ex: `transition: opacity 1s ease;`.

On peut placer plusieurs transitions séparées par des virgules. Ex: `transition: opacity 1s ease, background-color 1s ease;` ou `transition: all 1s ease;`.

```
.bouton {
    opacity: 1;
    transition: opacity 1s ease-in-out;
}
.bouton:hover {
    opacity: 0.5;
}
```

Les animations CSS

Les animations CSS permettent de définir des animations plus complexes avec plusieurs étapes. Pour définir une animation, vous devez spécifier les étapes de l'animation, la durée de l'animation, le nombre de répétitions et l'effet de l'animation. L'animation est utile pour les changements d'état plus complexes. Ex: une rotation de 360 degrés, un déplacement de gauche à droite, etc.

Pour définir l'animation, il faut d'abord définir les étapes de l'animation avec `@keyframes` puis appliquer l'animation à l'élément. On place les étapes de l'animation avec des pourcentages de 0 à 100%. Il est possible également d'utiliser les mots-clés `from` et `to` pour 0% et 100%.

Attention, dans la règle CSS `animation`, vous spécifiez le nom de l'animation sans les guillemets, la durée de l'animation, le nombre de répétitions et l'effet de l'animation.

```css
@keyframes rotation {
    0% {
        transform: rotate(0deg);
        background-color: red;
    }
    50% {
        background-color: orange;
    }
    100% {
        transform: rotate(360deg);
        background-color: yellow;
    }
}

.bouton {
    animation: rotation 2s infinite;
}
```

Propriétés de l'animation

- `animation-name` : Nom de l'animation défini avec `@keyframes`.
- `animation-duration` : Durée de l'animation.
- `animation-timing-function` : Effet de l'animation (linear, ease, ease-in, ease-out, ease-in-out, cubic-bezier).
- `animation-delay` : Délai avant le démarrage de l'animation.
- `animation-iteration-count` : Nombre de répétitions de l'animation (nombre ou infinite).
- `animation-direction` : Sens de l'animation (normal, reverse, alternate, alternate-reverse).
- `animation-fill-mode` : Style de l'animation avant et après l'exécution (none, forwards, backwards, both). Sert à définir le style de l'élément avant et après l'animation.

Propriétés animables

Toutes les propriétés CSS ne sont pas animables. Seules certaines propriétés peuvent être animées, comme les propriétés qui ont une valeur numérique.

- `background-color`, `color`
- `border-color`, `border-radius`, `border-width`, `border-spacing`
- `bottom`, `top`, `left`, `right`, `z-index`
- `font-size`, `font-weight`, `font-style`
- `height`, `width`
- `margin`, `padding`
- `opacity`, `visibility`
- `transform`, `rotate`, `scale`, `translate`

Les propriétés qui ne sont pas animables sont ignorées lors de l'animation. Ce sont généralement les propriétés qui ne sont pas numériques et qui ont des valeurs fixes en chaîne de caractères.

- `display`, `position`, `float`, `clear`

Contrôler les transitions CSS Pour contrôler les transitions CSS avec JavaScript,

Vous pouvez utiliser les événement `transitionstart` et `transitionend` en javaScript. Ces événements sont déclenchés au début et à la fin d'une transition. Ils permettent de contrôler le déroulement de la transition.

L'objectif généralement est d'attendre la fin d'une transition pour exécuter une fonction. Sans ce délai, la fonction serait exécutée immédiatement, ce qui pourrait entraîner des problèmes d'affichage.

Dans l'exemple suivant, nous allons ajouter une classe à un élément pour le rendre invisible, puis attendre la fin de la transition pour le supprimer du DOM.

La transiton démarre au moment où vous ajoutez la classe à l'élément.

Attention, les noms des événements sont en minuscules.

```css
.invisible {
    opacity: 0;
    transition: opacity 1s;
}
```

```javascript
let elementHTML = document.querySelector("section");

elementHTML.addEventListener("transitionend", function () {
    elementHTML.remove();
});

elementHTML.classList.add("invisible");
```

Contrôler les animations CSS

Il est possible de contrôler les animation de la même manière que les transitions. Pour cela, vous pouvez utiliser les événement `animationstart` et `animationend`. L'avantage des animations par rapport aux transitions est que vous pouvez définir des étapes intermédiaires pour l'animation.

L'animation démarre au moment où vous ajoutez la classe à l'élément.

Attention, les noms des événements sont en minuscules.

```
@keyframes disparition {
    0% {
        opacity: 1;
        background-color: red;
    }
    40% {
        opacity: 0.5;
        background-color: orange;
    }
    to {
        opacity: 0;
        background-color: yellow;
    }
}

.disparition {
    animation: disparition 1s;
}
```

```
let elementHTML = document.querySelector("section");
elementHTML.addEventListener("animationend", disparaitre);

function disparaitre() {
    // elementHTML.removeEventListener("animationend", disparaitre); // Optionnel
    elementHTML.classList.remove("disparition");
}

elementHTML.classList.add("disparition");
```

Redémarrer une animation CSS

Pour redémarrer une animation CSS, vous devez enlever la classe qui la déclenche, puis la remettre. Cela permet de forcer le navigateur à redémarrer l'animation. Cependant, il faut attendre que l'animation soit terminée pour la

redémarrer car en enlevant la classe, l'animation est stoppée, d'où l'intérêt de l'écoute de l'événement `animationend`.