IUT Bobigny
Université Sorbonne Paris Nord

UNIVERSITÉ SORBONNE PARIS NORD

BUT MMI 2ème année **Année 2025-2026**

TP n°6

Animations pour le Web

1 Animations en CSS

1.1 Les transitions

Les **transitions** CSS sont la solution la plus simple pour ajouter un peu d'animation dans les pages Web.

Le principe consiste à demander à une propriété CSS (largeur, hauteur etc.) de passer d'une valeur à une autre de façon progressive et sur une durée à préciser.

- `transition-property` permet de définir la propriété ciblée par la transition.
  Exemple : `transition-property : width;`
  Il est possible d'utiliser la valeur `all`.
- `transition-duration` permet de préciser la durée de la transition.
  Exemple : `transition-duration : 2s;`
- `transition-timing-function` prend une fonction comme valeur. Cette fonction permet de faire varier la vitesse de la transition en fonction du temps, par exemple, rapide puis lent puis rapide. Les valeurs possibles sont : `linear`, `ease` (valeur par défaut), `ease-in`, `ease-out` et `ease-in-out`. Il est possible de créer sa propre fonction avec la valeur `cubic-bezier(n,n,n,n)` où les paramètres `n` sont à personnaliser.
- `transition-delay` permet de différer le début de la transition d'une certaine durée.
  Exemple : `transition-delay : 1s;`

La propriété `transition` permet de regrouper toutes les caractéristiques de la transition dans une syntaxe raccourcie :
`transition: <property> <duration> <timing-function> <delay>;`

Souvent, le plus intéressant est d'appliquer une transition à une propriété de **transformation** :

```
div {
   transition : transform 2s;
}
```

CSS propose un certain nombre de transformations prêtes à l'emploi. Les valeurs possibles de la propriété `transform` sont :

- `translate(x,y)` où `x` et `y` sont des valeurs positives ou négatives exprimées dans une unité CSS valide (`px`, `%`, `vw`, `vh` etc.)
- `translateX(x)` pour une translation horizontale uniquement.
- `translateY(y)` pour une translation verticale uniquement.
- `scale(x,y)` pour définir une dilatation (changement d'échelle) selon les deux axes. Les dimensions de l'objet transformé sont multipliées par `x` horizontalement et par `y` verticalement.
- `scaleX(x)` pour une dilatation horizontale uniquement.
- `scaleY(y)` pour une dilatation verticale uniquement.

- `skew(x,y)` pour obtenir une distorsion de l'objet selon les deux axes. Les valeurs `x` et `y` sont des angles.
- `skewX(x)` selon l'axe horizontal uniquement.
- `skewY(y)` selon l'axe vertical uniquement.
- `rotate(x)` pour réaliser une rotation d'angle `x`.

En CSS, les angles peuvent être exprimés à l'aide de plusieures unités :

- En **degrés** (`deg`).
  Exemple `transform : rotate(60deg);`
- En **radians** (`rad`).
  Exemple `transform : rotate(2rad);`
- En **grade** (`rad`).
  Exemple `transform : rotate(100grad);`
- En **tours** (`turn`). Un tour complet correspond à la valeur `1turn`.
  Exemple `transform : rotate(0.5turn);`

Il est possible de définir sa propre transformation à l'aide d'une **matrice** de transformation. Pour une matrice homogène de la forme :

$$\begin{pmatrix} a & c & tx \\ b & d & ty \\ 0 & 0 & 1 \end{pmatrix}$$

Son écriture en CSS sera :

$$\text{matrix}(a, b, c, d, tx, ty)$$

Exemple : `transform : matrix(1,2,3,4,5,6);`

Pour finir, il est possible de modifier l'origine de la transformation (le centre de l'objet par défaut) à l'aide de la propriété `transform-origin`.

Syntaxe : `transform-origin : x-axis y-axis;`

Les valeurs possibles de `x-axis` sont :
- `left`
- `center`
- `right`
- un pourcentage `%`
- une longueur exprimée dans une unité CSS valide (`px`, `em` `vw`, `vh` etc.)

Les valeurs possibles de `y-axis` sont :
- `top`
- `center`
- `bottom`
- un pourcentage `%`
- une longueur exprimée dans une unité CSS valide (`px`, `em` `vw`, `vh` etc.)

1.2 Les animations

Les **animations** CSS permettent de réaliser des effets plus complexes.

La règle `@keyframes` permet de préciser le nom de l'animation et les différentes étapes qui la composent. Chaque étape correspond à un certain pourcentage d'avancement de l'animation. On peut définir autant d'étapes que l'on veut. Exemple :

```
@keyframes monAnimation {
  0% {
    width: 0;
    height: 0;
  }
  33% {
```

```
    width: 50px;
    height: 80px
  }
  66% {
    width: 100px;
  }
  100% {
    width: 100%;
    height: 200px;
  }
}
```

Seules les propriétés qui sont définies sur les étapes de début (0%) et de fin (100%) seront animées.

Notons que lorsque l'animation ne possède que 2 keyframes (forcément 0% et 100%) alors nous pouvons utiliser les alias `from` et `to` de la manière suivante :

```
@keyframes monAnimation {
  from {
    width: 0;
    height: 0;
  }
  to {
    width: 100%;
    height: 200px;
  }
}
```

Une fois l'animation ainsi définie, on peut l'utiliser sur un élément en précisant le nom et la durée :

```
div {
   animation-name : monAnimation;
   animation-duration : 5s;
}
```

Il est possible d'utiliser plusieurs animations sur un même élément à la suite :

```
div {
   animation-name : monAnimation1, monAnimation2;
   animation-duration : 5s, 1s;
}
```

Les autres propriétés applicables à une animation CSS sont :

- `animation-delay`. Elle permet de différer le début de la transition d'une certaine durée. Exemple : `animation-delay : 1s;`
- `animation-direction`. Elle précise l'ordre dans lequel l'animation doit se dérouler. Les valeurs possibles sont :
  — `normal` (valeur par défaut)
  — `reverse`
  — `alternate`
  — `alternate-reverse`
- `animation-fill-mode`. Elle permet de préciser quels styles doivent être appliqués à l'élément avant (pendant la durée du délai) et après l'animation. Les valeurs possibles sont :
  — `none` : valeur par défaut
  — `forwards` : on garde le style d'avant l'animation jusqu'au début de celle-ci (à la fin du délai) puis celui de la dernière keyframe de l'animation une fois terminée .

— `backwards` : On applique déjà le style de la première keyframe pendant la durée du délai puis on revient au style d'avant l'animation une fois terminée.
— `both` : On applique le style de la première keyframe pendant le délai et le style de la dernière keyframe à la fin.

- `animation-iteration-count`. Elle précise le nombre de fois que l'animation doit se répéter. Sa valeur est un entier ou `infinite` si elle se répète indéfiniment.
- `animation-play-state`. Elle permet de préciser l'état de l'animation. Les deux valeurs possibles sont `running` et `paused`.
- `animation-timing-function`. Elle permet de préciser une fonction d'accélération qui va faire varier la vitesse de l'animation en fonction du temps, par exemple, rapide puis lent puis rapide. Les valeurs possibles sont :
  — `linear`
  — `ease` (valeur par défaut)
  — `ease-in`
  — `ease-out`
  — `ease-in-out`
  — `cubic-bezier(n,n,n,n)`

Toutes les propriétés précédentes peuvent être définies à l'aide d'une syntaxe raccourcie en utilisant la propriété `animation` :

```
div {
  animation: monAnimation 3s ease-in-out 2s infinite alternate;
}
```

2 Animations en JavaScript

L'animation repose sur l'impression de mouvement. Il s'agit donc de modifier des éléments de la page web dans le temps. Pour cela, du code doit s'exécuter à intervalle de temps régulier.

En javascript, il existe plusieurs fonctions pour réaliser cela.

- `setTimeout(fonction, millisecondes)` exécute une fonction après un intervalle de temps donné
- `setInterval(fonction, millisecondes)` exécute une fonction de manière répétée selon un intervalle de temps donné.
- `clearInterval(var)` stop l'exécution répétée d'un `setInterval`

Ci-dessous un exemple ou javascript va permettre de changer la couleur de fond d'un élément toutes les 3 secondes :

```
<html>
  <head>
    <style rel="stylesheet">
      #carre {
        width:200px;
        height:200px;
      }
     </style>
  </head>
  <body>
    <div id="carre"></div>
  </body>
</html>
```

```
window.addEventListener('load',run,false);

function run(){
   let swap = true;
   setInterval(change,3000);

   function change() {
```

```
      var carre = document.getElementById("carre");
      console.log(swap);
      if (swap) {
         carre.style.backgroundColor = "red";
      } else {
         carre.style.backgroundColor = "blue";
      }
      swap = !swap;
   }
}
```

Il peut y avoir un problème lorsque qu'on exécute un nombre important d'appel à ces fonctions. Le navigateur ne les synchronise pas et cela peut entraîner une charge trop importante d'actions à réaliser dans un temps donné trop court.

Pour corriger ce problème, il existe maintenant la fonction `requestAnimationFrame(callback)` qui indique au navigateur qu'on souhaite exécuter une animation et demande que celui-ci exécute une fonction spécifique (`callback`) de mise à jour de l'animation, avant le prochain rafraîchissement à l'écran du navigateur.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple d'utilisation de requestAnimationFrame</title>
    <meta charset="utf-8" />
</head>
<body></body>
<script>
  let prev = performance.now();
  let compteur = 0;
  /* requestAnimationFrame s'appelle de façon récursive */
  requestAnimationFrame(function measure(time) {// time : temps écoulé depuis le
      chargement de la page
    document.body.insertAdjacentHTML("beforeEnd", Math.floor(time-prev) + " ");
    prev = time;
    if (compteur++ < 10) requestAnimationFrame(measure);
  })
</script>
</html>
```

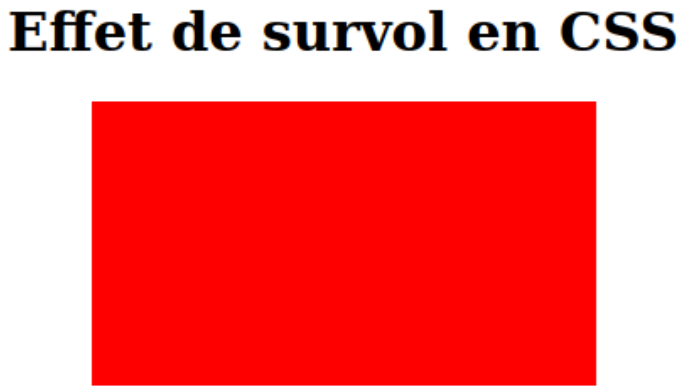
Notons que `requestAnimationFrame` ne s'exécute qu'une fois. Il faut donc que la fonction de callback prévoit un nouvel appel à cette fonction à chaque exécution (appel récursif).
Avec l'utilisation de cette fonction, le navigateur est capable de gérer et d'optimiser les appels à la fonction de rappel .

Vous trouverez avec ce document 2 archives contenant des exemples illustrant l'intérêt de faire appel à cette fonction :

- Dans `Chrono.zip`, on retrouve 3 compteurs de secondes codés sans puis avec `requestAnimationFrame`. Vous pourrez tester ce qu'il se passe pour chacun des compteurs lorsqu'on effectue un nombre important d'appui sur le bouton *start*.
- Dans `Exemples.zip`, on retrouve quelques animations d'un morceau de texte. Le code fourni fonctionne également pour l'animation d'images.

Exercice 1 :

Une image de votre choix est placée dans une page web. Elle se trouve derrière un élément opaque qui la cache.



Lorsque la souris survole la zone rectangulaire, l'élément opaque s'ouvre progressivement comme un rideau et laisse apparaître l'image.

Lorsque la souris quitte la zone, le rideau se referme.

Exercice 2 :

Vous devez réaliser quelques animations en CSS :

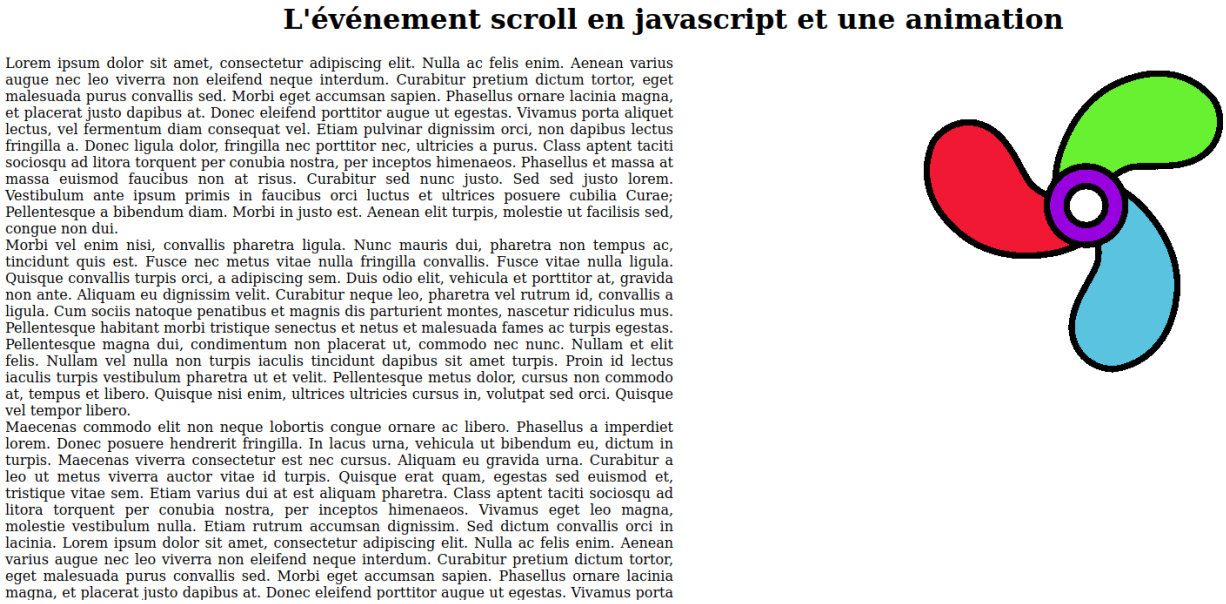
1. 3 points sont animés en changeant de taille de manière décalée pour illustrer un temps d'attente.

2. Des barres verticales changent de hauteur et produisent un effet de vague.
3. Une image tourne sur elle-même pour illustrer un temps d'attente.
4. Un texte sur fond vert apparaît caractère par caractère. Puis le fond devient rouge, l'élément tremble et le texte disparaît.
5. Un bouton de soumission change d'aspect lorsqu'il est cliqué.
6. Des éléments disposés en grille changent de forme avec un décalage les uns par rapport aux autres.
7. Du texte s'anime en faisant une vague.

Vous pouvez utiliser les images fournies dans l'archive `.zip`

Exercice 3 :

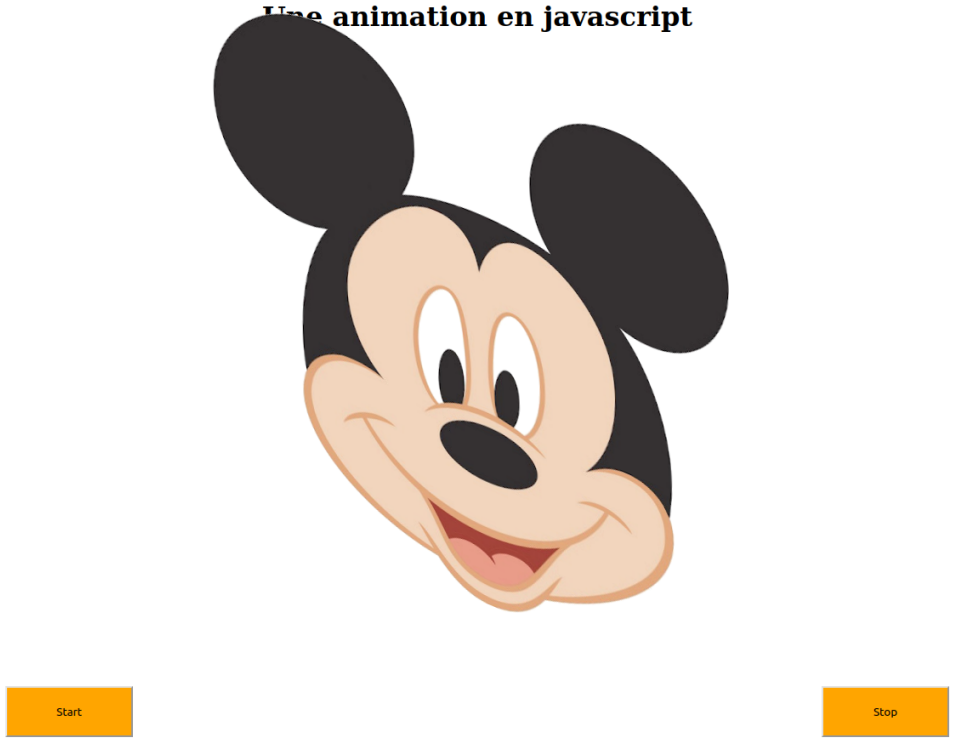
Lorsqu'un document avec un long contenu est scrollé, une image d'hélice placée en position fixe se met à tourner.



Vous pouvez utiliser l'image fournie dans l'archive.

Exercice 4 :

Il faut réaliser un document web contenant une image de votre choix et 2 boutons. Lorsqu'on clique sur le premier bouton, l'image se met à se déformer de façon aléatoire. Le résultat est obtenu en définissant une transition sur la propriété `transform`.
Ensuite, à intervalle de temps régulier et en utilisant javascript, les coefficients de la matrice que vous aurez affectée à la propriété `transform` changent aléatoirement de valeurs.



Exercice 5 :

Vous devez réaliser une bannière Web animée. Cette bannière a pour objectif de communiquer autour d'un produit, d'une marque, d'un événement, d'une cause humanitaire etc.

- Le travail doit être individuel. Je ne veux pas voir 2 bannières identiques ou sur le même sujet.
- Les dimensions de la bannière doivent être de 728 × 90 px (dimensions standard pour une bannière horizontale).
- Elle doit comporter au moins un texte animé et une image animée. L'animation doit être obtenue à l'aide de CSS ou Javascript (ou les deux), pas d'image gif autorisée