Transitions, animations, transformations

Rédigé par: Jean-Luc Colson
Date première rédaction: Janvier 2020

SUIVI DES MODIFICATIONS A LA FICHE

Série	Date	Page(s) modifiée(s)	Raison

Contents

Créer des transitions en CSS

Jusqu'à présent, à chaque fois qu'on définissait le comportement d'une propriété CSS pour un élément, la valeur définie s'appliquait directement.

Les transitions CSS vont nous permettre de modifier la valeur d'une propriété CSS de manière fluide et selon une durée que l'on va pouvoir définir. On va donc pouvoir définir deux valeurs pour une propriété (une première valeur de départ ou valeur par défaut et une seconde valeur d'arrivée) et faire en sorte que la valeur change progressivement de la valeur de départ à la valeur d'arrivée.

On va ainsi par exemple pouvoir changer progressivement la couleur des textes de nos éléments ou modifier la taille d'un élément, etc.
Pour créer des transitions en CSS, nous allons pouvoir utiliser les différentes propriétés de type transition-* ou la propriété raccourcie transition.

Dans cette nouvelle leçon, nous allons commencer par étudier les propriétés complètes qui sont les suivantes :

- transition-property ;
- transition-duration ;
- transition-timing-function ;
- transition-delay.

Nous terminerons avec la création d'une transition complète en utilisant la propriété raccourcie transition.

Le fonctionnement d'une transition

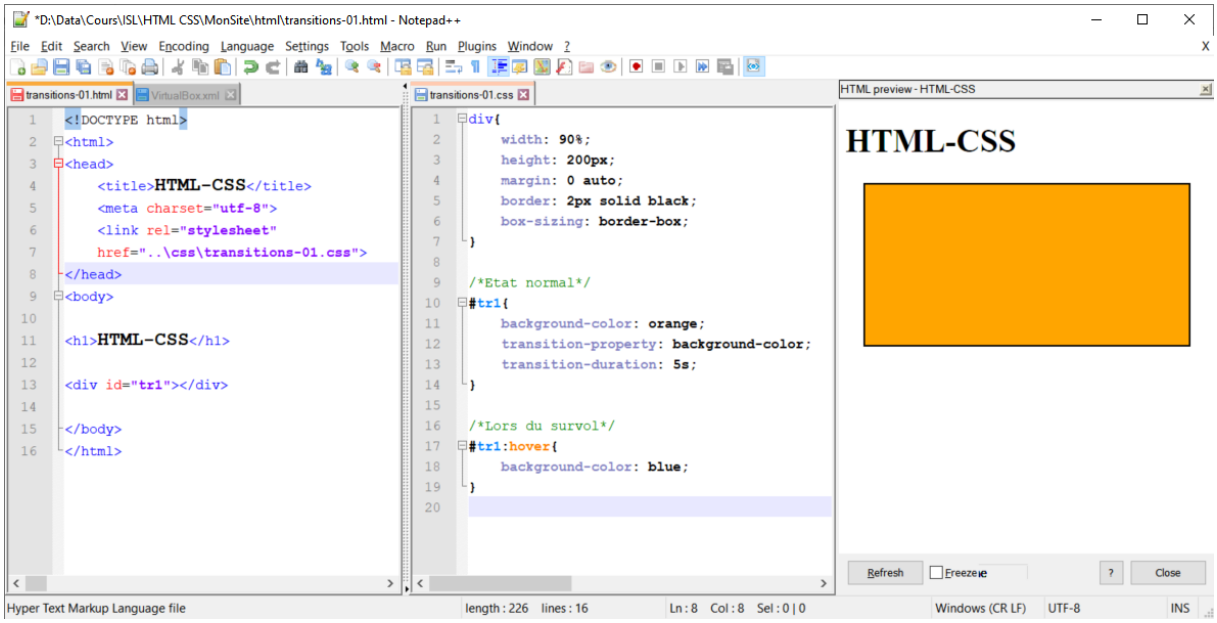
Le principe de base d'une transition en CSS est de modifier progressivement la valeur d'une propriété. Pour cela, nous allons devoir indiquer deux valeurs pour la propriété pour laquelle on souhaite créer une transition.

Le premier souci ici est qu'on ne va pas pouvoir passer deux valeurs pour une même propriété avec un même sélecteur CSS pour créer une transition, car en faisant cela seule la dernière valeur déclarée serait lue.

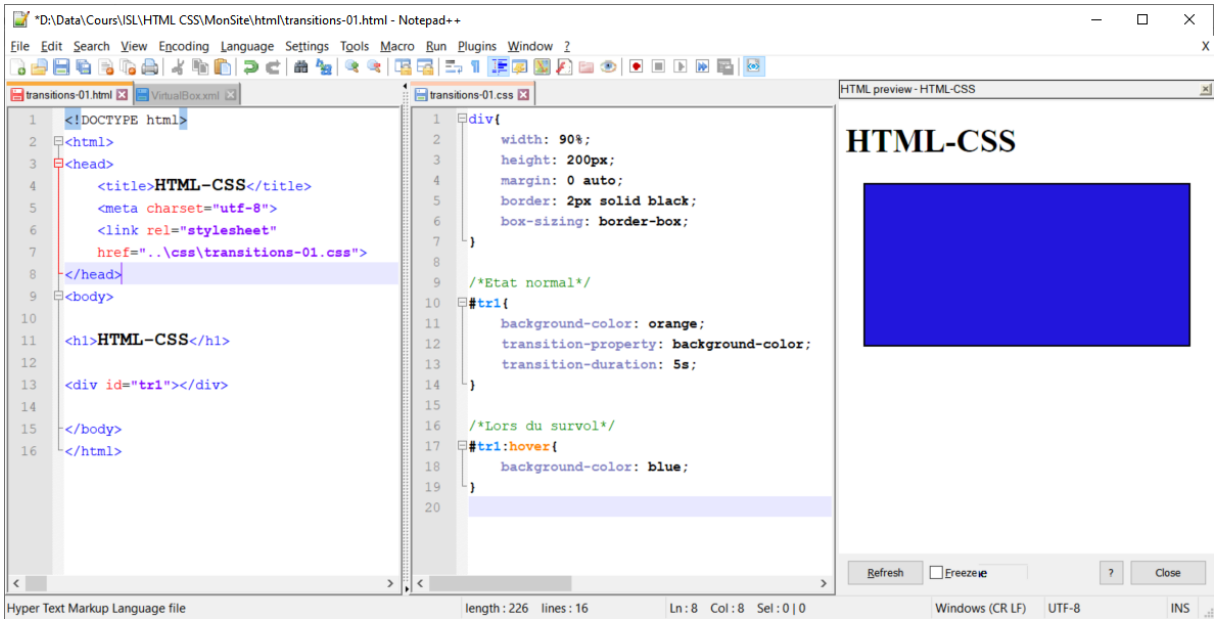
Nous avons ici deux solutions pour contourner cette contrainte. La première qui va nous permettre d'exploiter tout le potentiel des animations va être d'utiliser un langage dynamique comme le JavaScript qui peut mettre à jour le nom d'un attribut class par exemple en fonction de certains critères.

Comme nous n'avons pas étudié le JavaScript, nous allons mettre de côté cette première solution. La deuxième méthode va être d'utiliser les transitions avec les pseudo-classes, c'est-à-dire lors d'un changement d'état d'un élément HTML.

Nous allons par exemple pouvoir changer la couleur de fond d'un élément div lorsqu'un utilisateur passe sa souris dessus comme ci-dessous :



Au passage de la souris, la couleur de fond change progressivement vers le bleu en 5 secondes, et retour à l'orange lorsque la souris quitte le div:



Ici, on a utilisé deux propriétés pour créer une transition simple : la propriété transition-property qui sert à définir la ou les propriété(s) dont la valeur doit être modifiée progressivement et la propriété transition-duration qui indique le temps que va mettre la propriété à passer de sa valeur de départ à la valeur d'arrivée.

Comme vous pouvez le constater en observant le code CSS, on appliquer ces propriétés de transition à notre div dans son état normal et on définit deux couleurs de fond : orange dans l'état normal et bleu lorsque le div est survolé.

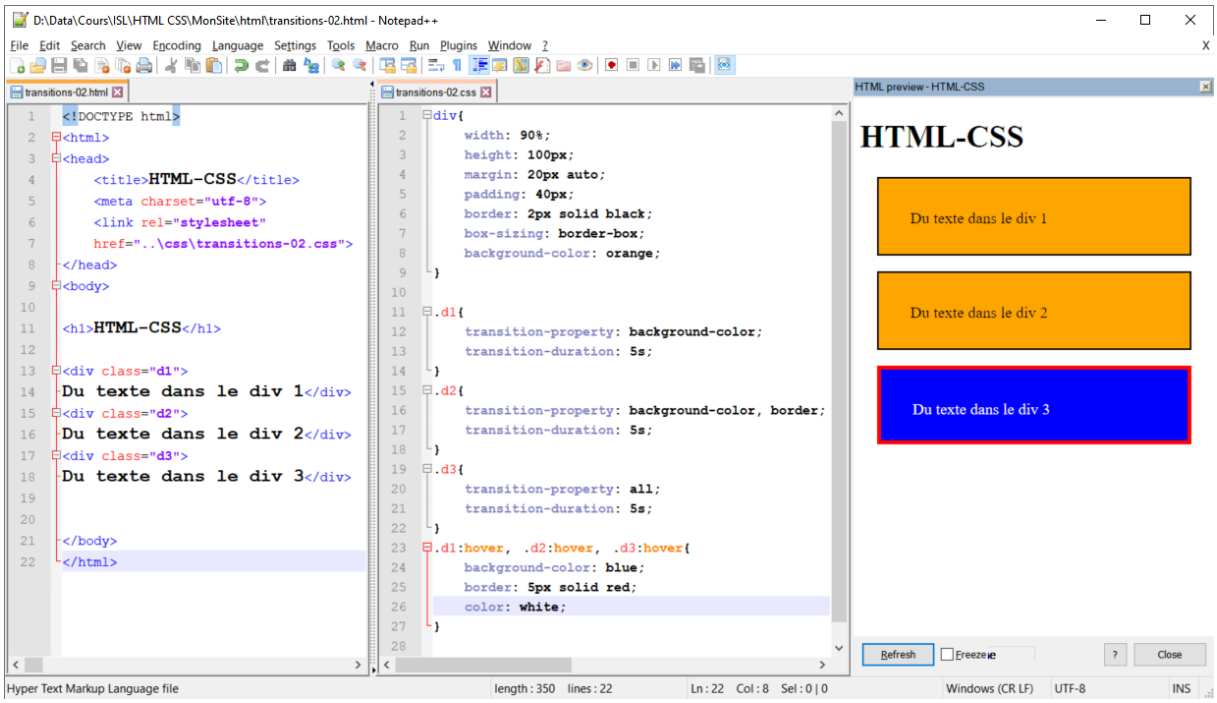
Lorsqu'on survole le div, sa couleur de fond va donc changer progressivement de l'orange vers le bleu sur une durée de 5 secondes. Notez que par défaut la transition se fait dans les deux sens : lorsqu'on sort du div, sa couleur de fond retourne progressivement vers l'orange.

Notez finalement qu'on ne va pas pouvoir créer des transitions avec toutes les propriétés car le concept de transitions est assez récent en CSS et donc le support n'est pas encore parfait. La plupart des propriétés vont tout de même pouvoir être animées.

La propriété CSS transition-property

La propriété transition-property va nous permettre de définir quelles propriétés vont être les cibles de nos transitions, c'est-à-dire quelles sont les propriétés dont la valeur va devoir changer progressivement.

On va pouvoir indiquer en valeur de transition-property soit le nom d'une propriété CSS pour laquelle on veut créer une transition, soit le nom de plusieurs propriétés CSS qui devront alors être séparées par des virgules, soit le mot clef all qui signifie que toutes les propriétés vont pouvoir être sujettes aux transitions (sous réserve qu'on indique ensuite une autre valeur d'arrivée pour créer la transition bien évidemment).



Dans cet exemple, on crée trois transitions pour chacun de nos trois div. Ici, on définit de nouvelles valeurs pour les propriétés color ; background-color et border de nos div lors de l'état hover.

Les transitions vont donc se faire lorsque l'utilisateur va passer sa souris sur un des div. Chacune des 3 transitions va durer 5 secondes.

Pour notre premier div, on applique la transition à la propriété background-color uniquement.

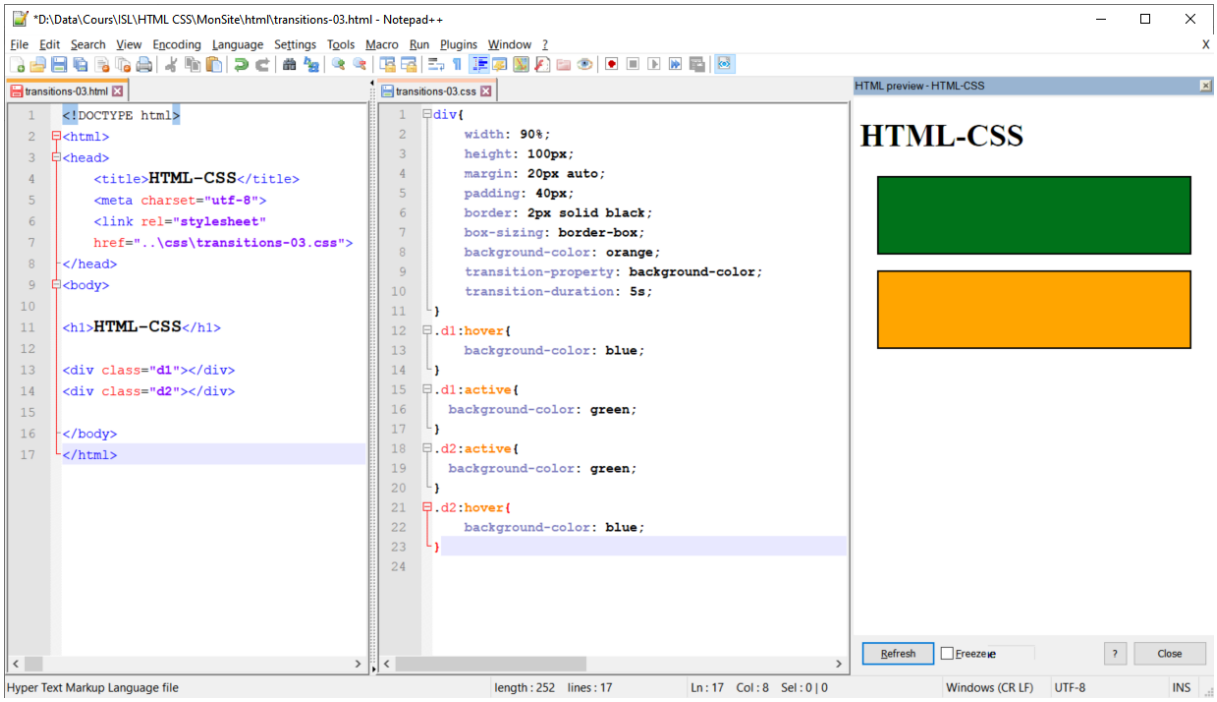
Pour notre deuxième div, on applique la transition aux propriétés background-color et border.

Pour notre troisième div, on applique une transition à toutes les propriétés qui ont plusieurs valeurs définies pour différents états de l'élément.

Notez que je n'ai pas défini de couleur de départ avec color. En effet, il n'est pas toujours strictement indispensable de définir des valeurs de départ pour les propriétés pour lesquelles on définit des transitions puisque la plupart des propriétés ont des valeurs par défaut. Pour color, la transition va donc se faire entre la valeur par défaut (noire) et la valeur d'arrivée spécifiée.

Ici, vous devez également bien comprendre qu'une transition ne va se déclencher que lors d'un changement d'état d'un élément. Ainsi, il est tout à fait possible de définir plus de deux valeurs pour une propriété à laquelle on souhaite appliquer une transition puisqu'on va pouvoir définir une valeur pour chaque état de l'élément.

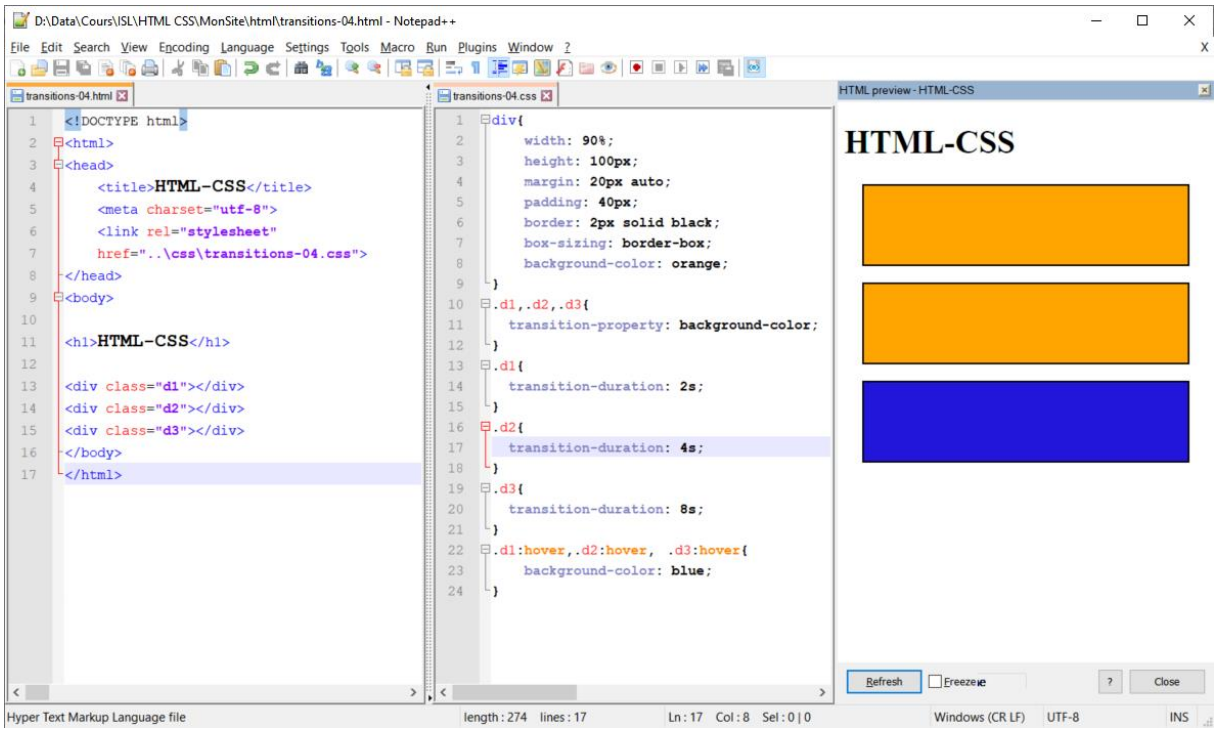
Dans le cas où plusieurs changements d'état sont provoqués en même temps, la transition se fera vers la valeur de l'état déclaré en dernier en CSS. Par exemple, on peut définir une transition sur une couleur de fond en précisant une valeur d'arrivée pour l'état hover (survol) et l'état active (cliqué) :



La propriété CSS transition-duration

La propriété transition-duration va nous permettre de définir le temps que vont mettre les propriétés passées à transition-property pour passer d'une valeur de départ à une valeur d'arrivée. Nous allons pouvoir lui passer un nombre de secondes en valeur.

La valeur par défaut de transition-duration est 0s ce qui signifie que le changement de valeur des propriétés concernées par la transition sera immédiat.




```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet"
    href="..\css\transitions-04.css">
</head>
<body>

<h1>HTML-CSS</h1>

<div class="d1"></div>
<div class="d2"></div>
<div class="d3"></div>
</body>
</html>
```

```
div{
    width: 90%;
    height: 100px;
    margin: 20px auto;
    padding: 40px;
    border: 2px solid black;
    box-sizing: border-box;
    background-color: orange;
}
.d1,.d2,.d3{
  transition-property: background-color;
}
.d1{
  transition-duration: 2s;
}
.d2{
  transition-duration: 4s;
}
.d3{
  transition-duration: 8s;
}
.d1:hover,.d2:hover, .d3:hover{
   background-color: blue;
}
```

Ici, on applique une transition sur la couleur de fond de nos éléments HTML div lors du passage de la souris sur l'un deux. Chaque transition a une durée différente définie avec transition-duration ce qui signifie que la couleur de fond mettra plus ou moins de temps à passer de sa couleur de départ à sa couleur d'arrivée selon le div.

La propriété CSS transition-timing-function

La propriété transition-timing-function va nous permettre de choisir la vitesse de la transition au sein de celle-ci. Nous allons ainsi pouvoir créer des transitions totalement linéaires ou, au contraire, créer des transitions qui vont s'accélérer ou ralentir au milieu. Nous allons pouvoir passer les valeurs suivantes à cette propriété :

- ease : valeur par défaut. Permet de créer une transition relativement lente au début puis qui s'accélère au milieu et qui se termine lentement ;
- linear : permet de créer une transition totalement linéaire c'est-à-dire qui va aller à la même vitesse du début à la fin ;
- ease-in : permet de créer une transition avec un départ lent puis qui s'accélère ensuite ;

- ease-out : permet de créer une transition qui va ralentir à la fin ;
- ease-in-out : permet de créer une transition lente au début puis qui s'accélère au milieu et qui se termine lentement. Ressemble fortement à transition-timing-function : ease mais démarre plus lentement ;
- cubic-bezier(x1,y1,x2,y2) : sert à créer une transition à la vitesse totalement personnalisée en renseignant une courbe de Bézier.

Notre but n'est ici bien évidemment pas de faire un cours sur les courbes de Bézier. Vous pouvez simplement retenir les équivalents mot-clef/Bézier suivants qui peuvent se révéler utiles :

- transition-timing-function : ease est équivalent à transition-timing-function : cubic-bezier(0.25, 0.1, 0.25, 1) ;
- transition-timing-function : ease-in est équivalent à transition-timing-function : cubic-bezier(0.42, 0, 1, 1) ;
- transition-timing-function : ease-out est équivalent à transition-timing-function : cubic-bezier(0, 0, 0.58, 1) ;
- transition-timing-function : ease-in-out est équivalent à transition-timing-function : cubic-bezier(0.42, 0, 0.58, 1).

D:\Data\Cours\ISL\HTML CSS\MonSite\html\transitions-05.html - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

transitions-05.html

```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet"
    href="..\css\transitions-05.css">
</head>
<body>

<h1>HTML-CSS</h1>

<div class="d1">
transition-timing-function:
ease</div>
<div class="d2">
transition-timing-function:
ease-in</div>
<div class="d3">
transition-timing-function:
ease-out</div>
<div class="d4">
transition-timing-function:
ease-in-out</div>
<div class="d5">
transition-timing-function:
linear</div>
<div class="d6">
transition-timing-function:
cubic-bezier(0.8, 0, 1, 1)
</div>

</body>
</html>
```

transitions-05.css

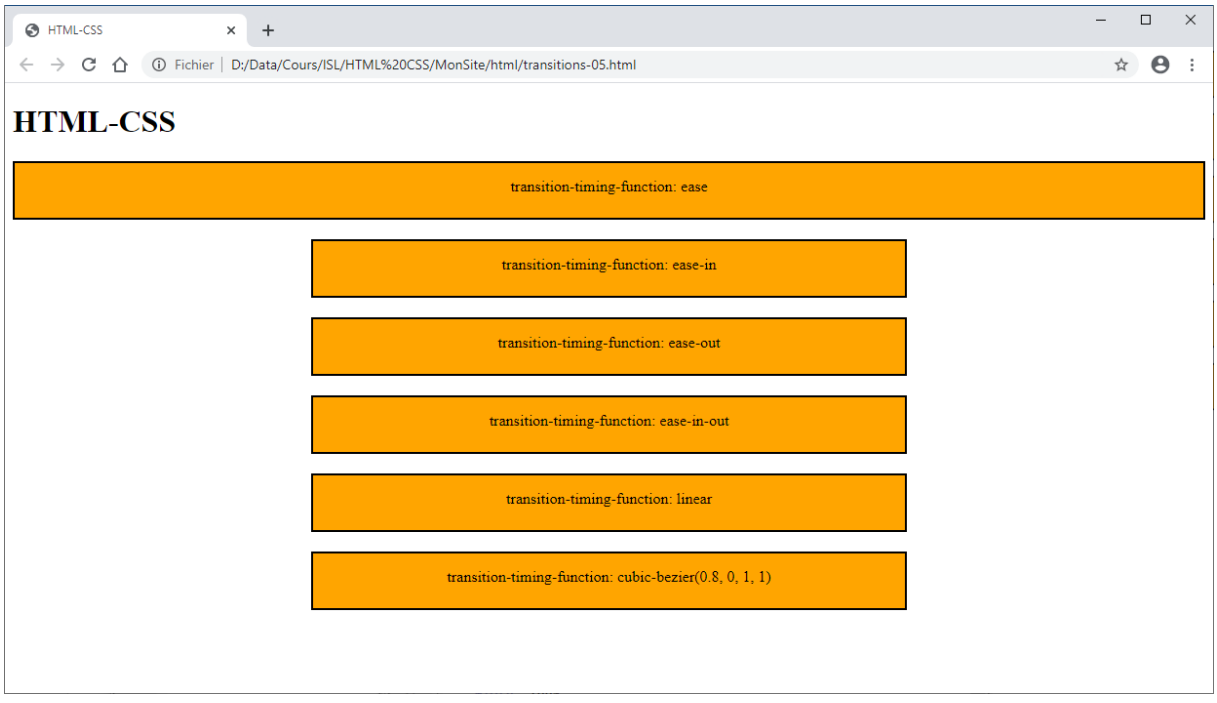
```
div{
    width: 50%;
    height: 60px;
    margin: 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
    text-align: center;
    padding: 15px;
}

.d1, .d2, .d3, .d4, .d5, .d6{
    background-color: orange;
    transition-property: width;
    transition-duration: 2s;
}

.d1{
    transition-timing-function: ease;
}
.d2{
    transition-timing-function: ease-in;
}
.d3{
    transition-timing-function: ease-out;
}
.d4{
    transition-timing-function: ease-in-out;
}
.d5{
    transition-timing-function: linear;
}
.d6{
    transition-timing-function: cubic-bezier(0.6, 0, 1, 1);
}

.d1:hover, .d2:hover, .d3:hover, .d4:hover, .d5:hover, .d6:hover{
    width: 100%;
}
```

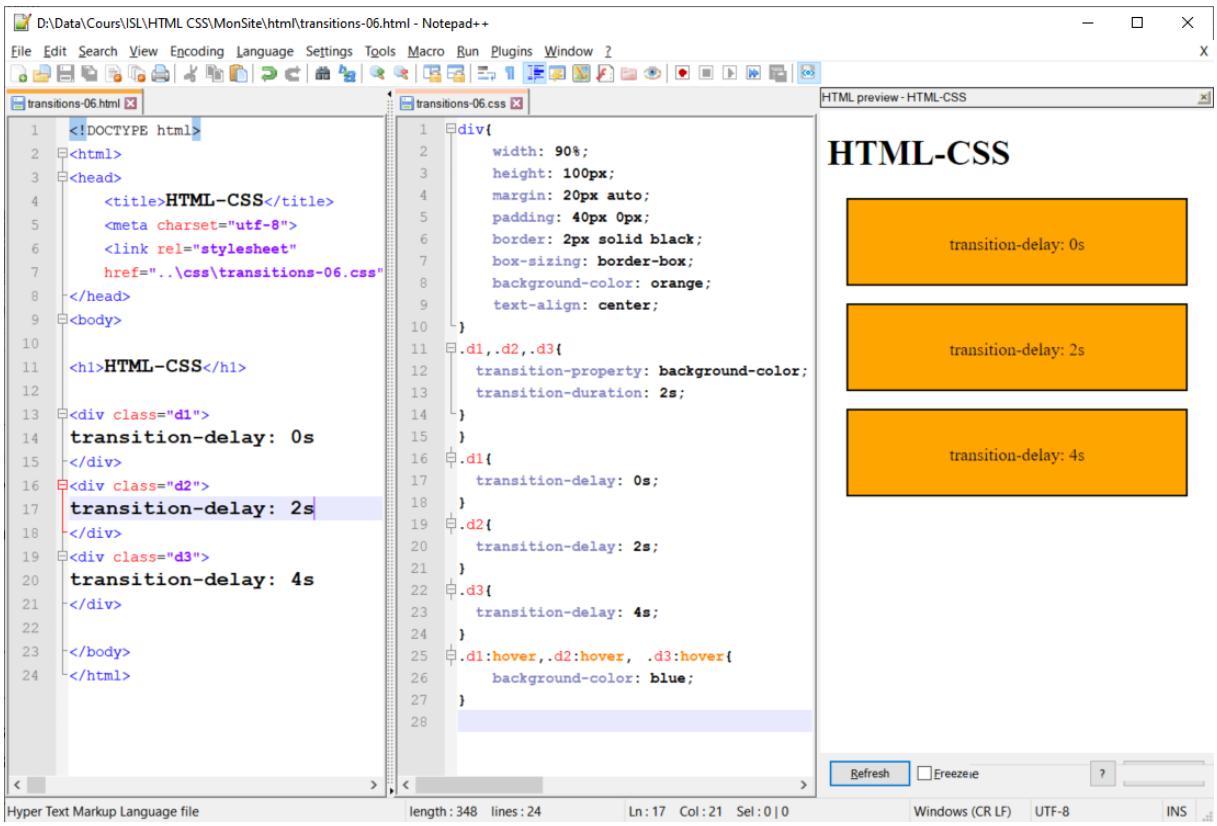
Hyper Text Markup Language file | length : 603 | lines : 34 | Ln : 17 Col : 24 Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS



La propriété CSS transition-delay

La propriété transition-delay va nous permettre de définir quand la transition doit commencer à partir du moment où la nouvelle valeur est passée aux propriétés concernées par la transition. On va pouvoir lui passer une valeur en secondes.

La valeur par défaut est 0s (la transition se lance dès qu'une nouvelle valeur est définie pour une propriété à laquelle on a appliqué la transition).



La propriété CSS transition

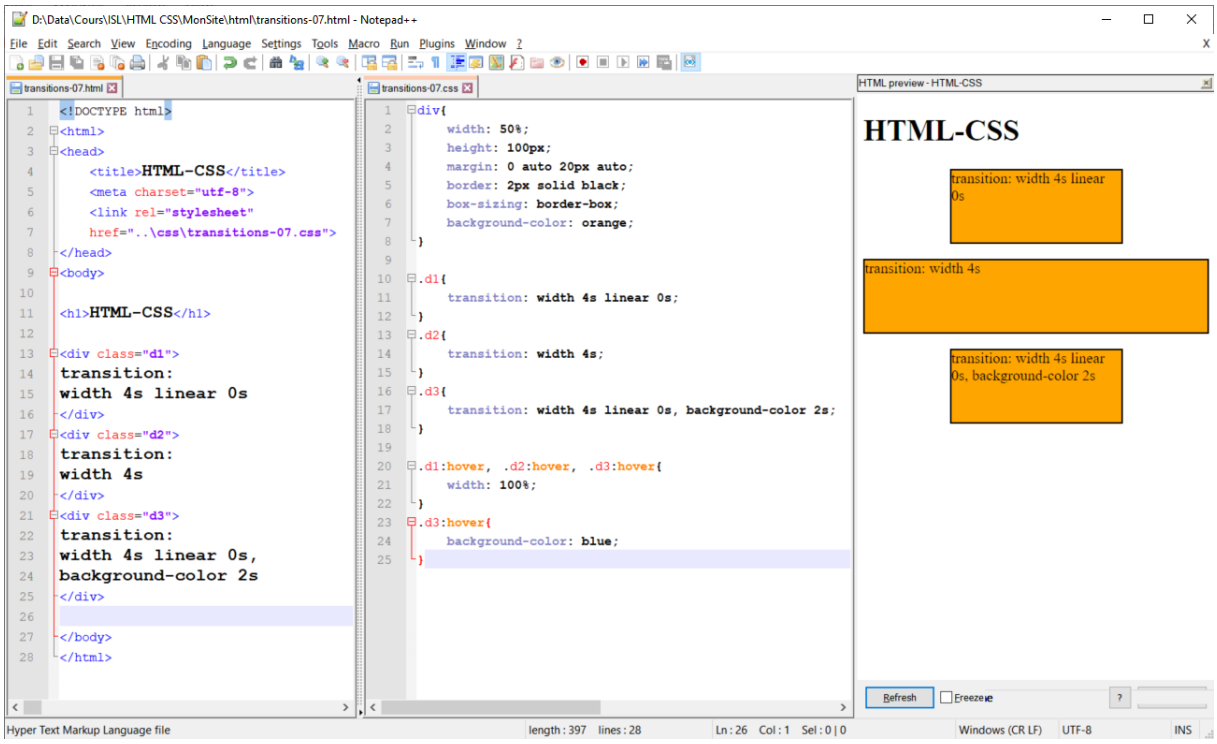
La propriété CSS transition est la notation raccourcie des quatre propriétés étudiées précédemment. On va pouvoir lui passer les différentes valeurs des propriétés précédentes à la suite pour créer une transition complète.

La première durée renseignée dans transition définira la durée de la transition. Je vous conseille de suivre l'ordre de déclaration des valeurs suivant pour être sûr que transition fonctionne bien :

1. La valeur relative à transition-property ;
2. La valeur relative à transition-duration ;
3. La valeur relative à transition-timing-function ;
4. La valeur relative à transition-delay.

A noter que seules les valeurs relatives aux propriétés transition-property et transition-duration doivent être obligatoirement renseignées pour créer une transition visible. Les valeurs relatives aux propriétés transition-timing-function et transition-delay sont facultatives et si rien n'est renseigné les valeurs par défaut seront utilisées.

Notez également que pour créer plusieurs transitions pour plusieurs propriétés différentes avec transition il va suffire de séparer les différentes déclarations par une virgule.



Ici, on renseigne toutes les valeurs pour notre première transition dans la propriété transition. Pour notre deuxième transition, en revanche, on omet les deux dernières valeurs. Ce seront donc les valeurs par défaut ease et 0s qui seront utilisées.

Finalement, nous créons deux transitions dans notre dernier div. Pour faire cela, on se contente de séparer les différentes déclarations pour chacune de nos transitions par une virgule dans notre propriété transition.

Transition et reverse transition

Par défaut, lorsqu'on crée une transition entre les valeurs d'une ou de plusieurs propriétés, la transition va se faire dans les deux sens :

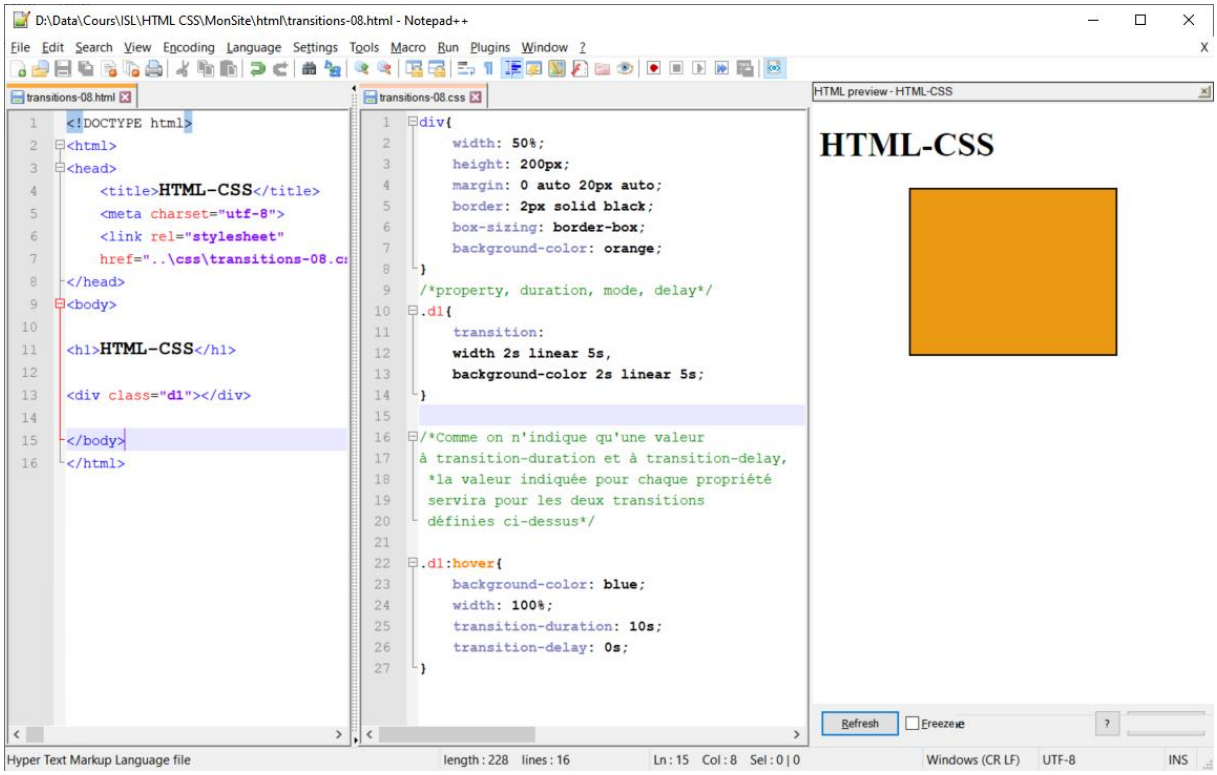
1. A partir de la valeur de départ vers la valeur d'arrivée lors du changement d'état d'un élément ;

2. A partir de la valeur actuelle vers la valeur de départ lorsque l'élément retourne dans son état initial.

Par exemple, si on crée une transition qui doit se déclencher lors du passage de la souris sur un élément, dès que l'on va déplacer notre souris en dehors de l'élément les propriétés concernées par la transition vont reprendre petit à petit leurs valeurs initiales en effectuant une transition inversée vers leurs valeurs initiales plutôt que d'être redéfinie brutalement sur leurs valeurs de départ.

Nous allons pouvoir également pouvoir gérer la transition de retour et notamment la durée de cette transition et son délai. Pour faire cela, nous allons devoir préciser en plus de nos valeurs de transition classiques des valeurs de transition dans l'état qui va déclencher la transition (à l'intérieur d'un :hover par exemple pour une transition qui doit démarre lorsque l'utilisateur passe sa souris sur un élément).

Attention ici : les valeurs précisées dans cet état seront utilisées pour la transition de départ et ce sont les valeurs précisées au départ qui seront utilisées pour la transition de retour. Regardez l'exemple suivant pour bien comprendre :




```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet"
    href="..\css\transitions-08.c
</head>
<body>

<h1>HTML-CSS</h1>

<div class="d1"></div>

</body>
</html>
```

```
div{
    width: 50%;
    height: 200px;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
    background-color: orange;
}
 /*property, duration, mode, delay*/
.d1{
    transition:
    width 2s linear 5s,
    background-color 2s linear 5s;
}

/*Comme on n'indique qu'une valeur
à transition-duration et à transition-delay,
 *la valeur indiquée pour chaque propriété
 servira pour les deux transitions
 définies ci-dessus*/

.d1:hover{
    background-color: blue;
    width: 100%;
    transition-duration: 10s;
    transition-delay: 0s;
}
```

Ici, on crée une transition sur la couleur de fond et la taille de notre div class="d1" qui doit démarrer lorsqu'un utilisateur passe sa souris sur notre div.

Cette première transition ici va durer 10 secondes et avoir un délai de 0s. Dès que l'utilisateur sort sa souris de l'élément, les valeurs de nos propriétés vont revenir à leur état normal en effectuant une transition de retour. La transition de retour va durer 2 secondes et va démarrer après un délai de 5 secondes.

La relation entre les valeurs et les transitions peut paraitre contre intuitif à priori, donc faites bien attention à bien retenir cela !

Créer des animations en CSS

Les animations vont, comme les transitions, nous nous permettre de modifier la valeur d'une propriété progressivement mais en utilisant cette fois-ci des keyframes.

Nous allons pouvoir gérer le comportement des animations en CSS en définissant la durée d'animation, le nombre de répétition et le comportement de répétition.

De manière similaire aux animations, toutes les propriétés ne vont pas pouvoir être animées. Cependant, la grande majorité des propriétés courantes vont pouvoir l'être.

Pour définir une animation en CSS nous allons pouvoir utiliser les propriétés de type animation-* ou la notation raccourcie animation avec une règle @keyframes qui va contenir les propriétés à animer et leurs valeurs.

Différence entre animations et transitions en CSS

Les transitions et les animations permettent toutes les deux de modifier la valeur de propriétés de manière progressive, au cours du temps. Cependant, la façon dont vont procéder les transitions et les animations pour arriver à cela ne va pas être la même.

La grande différence entre les transitions et les animations en CSS est que les animations nous laissent à la fois une plus grande liberté et un plus grand contrôle sur le déclenchement et la progression du changement de valeur des propriétés animées.

En effet, dans le cas d'une transition, nous ne pouvons que préciser une valeur de départ et une valeur d'arrivée pour les propriétés pour lesquelles nous souhaitons créer notre transition et n'avons pas véritablement de contrôle précis sur la transition en soi tandis que dans le cas d'une animation nous allons pouvoir indiquer de manière explicite comment la « transition » entre les différentes valeurs doit se passer et pouvoir préciser différentes valeurs intermédiaires.

En cela, les animations offrent davantage de contrôle sur le changement de valeurs des propriétés concernées par l'animation puisqu'on va pouvoir contrôler ce changement de valeur dans son ensemble. Elles vont donc notre choix de prédilection lorsqu'on voudra créer des effets plus complexes ou précis.

De plus, nous n'allons plus devoir attendre un changement d'état d'un élément pour modifier la valeur d'une de ses propriétés avec les animations. En effet, nous allons pouvoir lancer une animation dès le chargement de la page ou selon un autre évènement.

La règle CSS @keyframes

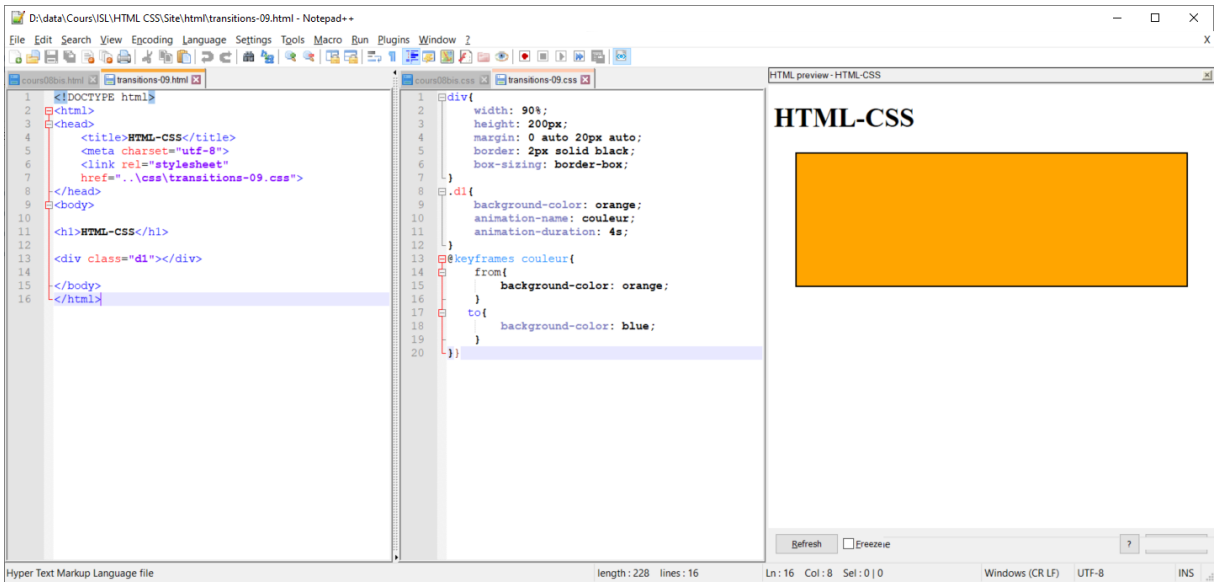
La règle @keyframes va nous permettre d'indiquer quelles propriétés doivent être animées et comment elles doivent l'être. Nous allons pouvoir dans notre règle @keyframes préciser différentes valeurs auxquelles les propriétés animées doivent parvenir à certains stades de l'animation.

La règle @keyframes va donc nous permettre de définir différents stades ou étapes pour notre animation et c'est ce qui va nous permettre d'avoir un contrôle total sur la progression de l'animation.

Nous allons toujours devoir donner un nom à une règle @keyframes qu'on va ensuite réutiliser dans notre animation pour lui indiquer les propriétés à animer et comment les animer. Dans notre règle @keyframes, nous allons tout simplement renseigner les propriétés à animer et les différentes valeurs qu'elles doivent atteindre à certains moments de l'animation.

Je vous rappelle ici que pour créer une animation fonctionnelle en CSS, nous allons d'une part devoir définir notre règle @keyframes et d'autre part utiliser les propriétés de type animation-* ou la notation raccourcie animation pour définir les caractéristiques propres à notre animation (nom, durée, etc.).

Regardez plutôt l'exemple suivant pour bien comprendre (l'animation va se lancer immédiatement, n'hésitez pas à la relancer en cliquant sur F5 pour rafraîchir la page) :



Ici, vous pouvez déjà remarquer une différence intéressante entre les animations et les transitions : une fois l'animation terminée, les propriétés animées retrouvent immédiatement leur valeur « normale ».

Regardons maintenant en détail notre règle @keyframes. Ici, on lui donne le nom couleur et on précise dedans les propriétés qui vont être animées (en l'occurrence la propriété background-color).

Plus précisément, on indique ici la valeur de départ pour notre ou nos propriété(s) que l'on souhaite animer à l'intérieur d'un bloc from{} qui signifie « à partir de » et la valeur d'arrivée pour notre ou nos propriété(s) que l'on souhaite animer à l'intérieur d'un bloc to{} qui signifie « vers ».

A ce niveau, il y a une chose que vous devez bien comprendre : la valeur que l'on précise dans le bloc from{} est bien la valeur de départ de l'animation, c'est-à-dire la valeur avec laquelle la propriété doit

démarrer son animation et non pas sa valeur « normale » c'est-à-dire sa valeur en dehors du temps de l'animation (avant le lancement de l'animation et après sa fin).

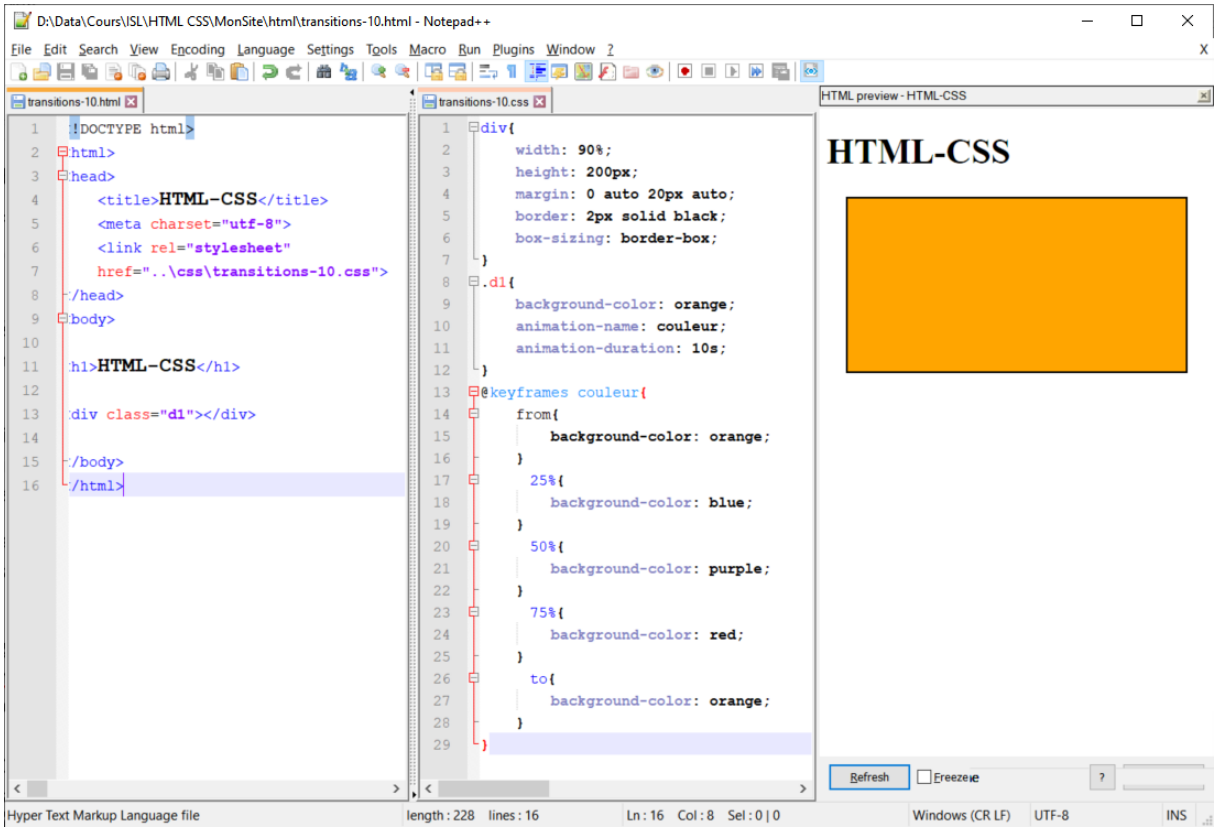
C'est pour cela que j'ai également précisé une couleur de fond en dehors de ma règle @keyframes afin de préciser la valeur que doit avoir ma propriété lorsqu'elle n'est pas animée.

Une fois notre règle @keyframes définie, nous allons devoir la lier ou l'attribuer à une animation. Pour cela, nous allons également devoir déclarer une animation et définir son comportement.

C'est ce que j'ai fait ici en déclarant deux propriétés deux propriétés animation-name à laquelle j'ai passé notre règle @keyframes et animation-duration qui me permet de définir la durée de l'animation. Par défaut, l'animation ne va s'exécuter qu'une fois.

Dans l'exemple précédent, utiliser une animation n'a aucun intérêt par rapport à une simple transition car cette animation est très simple. Cependant, rappelez-vous qu'un des grands avantages des animations par rapport aux transitions est qu'on va pouvoir dans notre règle @keyframes préciser différentes valeurs que nos propriétés devront atteindre à un moment choisi de l'animation.

On va ainsi entre autres très facilement pouvoir créer une boucle sur les valeurs d'une propriété pendant une animation comme dans l'exemple suivant :




```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet"
    href="..\css\transitions-10.css">
</head>
<body>

<h1>HTML-CSS</h1>

<div class="d1"></div>

</body>
</html>
```

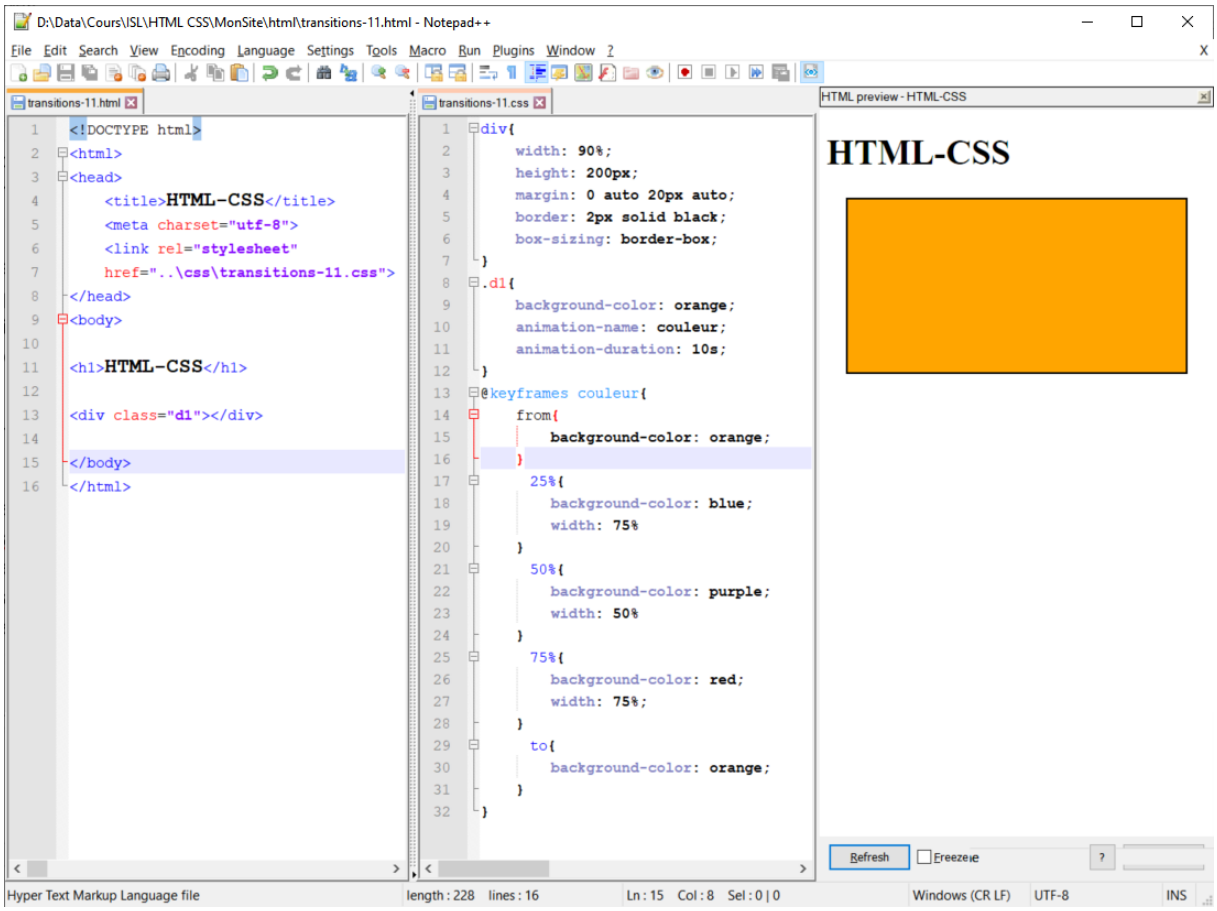
```
div{
    width: 90%;
    height: 200px;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1{
    background-color: orange;
    animation-name: couleur;
    animation-duration: 10s;
}
@keyframes couleur{
    from{
        background-color: orange;
    }
    25%{
        background-color: blue;
    }
    50%{
        background-color: purple;
    }
    75%{
        background-color: red;
    }
    to{
        background-color: orange;
    }
}
```

Ici, en plus de mes mots clefs from et to, j'ai indiqué des pourcentages dans ma règle @keyframes. Ces pourcentages correspondent à l'état d'avancement de l'animation. 50% marque donc le milieu de l'animation, c'est-à-dire au bout de 5 secondes pour une animation qui doit durer 10 secondes ;

25% correspond à 25% du temps total de l'animation, soit 2,5 secondes pour une animation de 10 secondes et etc. Ensuite, on indique la valeur que doit avoir notre propriété animée pour chaque niveau d'avancement de l'animation défini.

Le grand avantage des animations encore une fois est que nous allons pouvoir définir autant d'étapes que l'on souhaite pour contrôler plus ou moins le l'avancement de notre animation plus. J'attribue ici la même valeur à ma propriété background-color au début et en fin d'animation pour donner l'impression que la couleur boucle sur elle-même.

Nous allons de plus pouvoir préciser plusieurs propriétés à animer d'un coup et leur attribuer différentes valeurs à différents moments de l'animation au sein d'une même règle @keyframes :



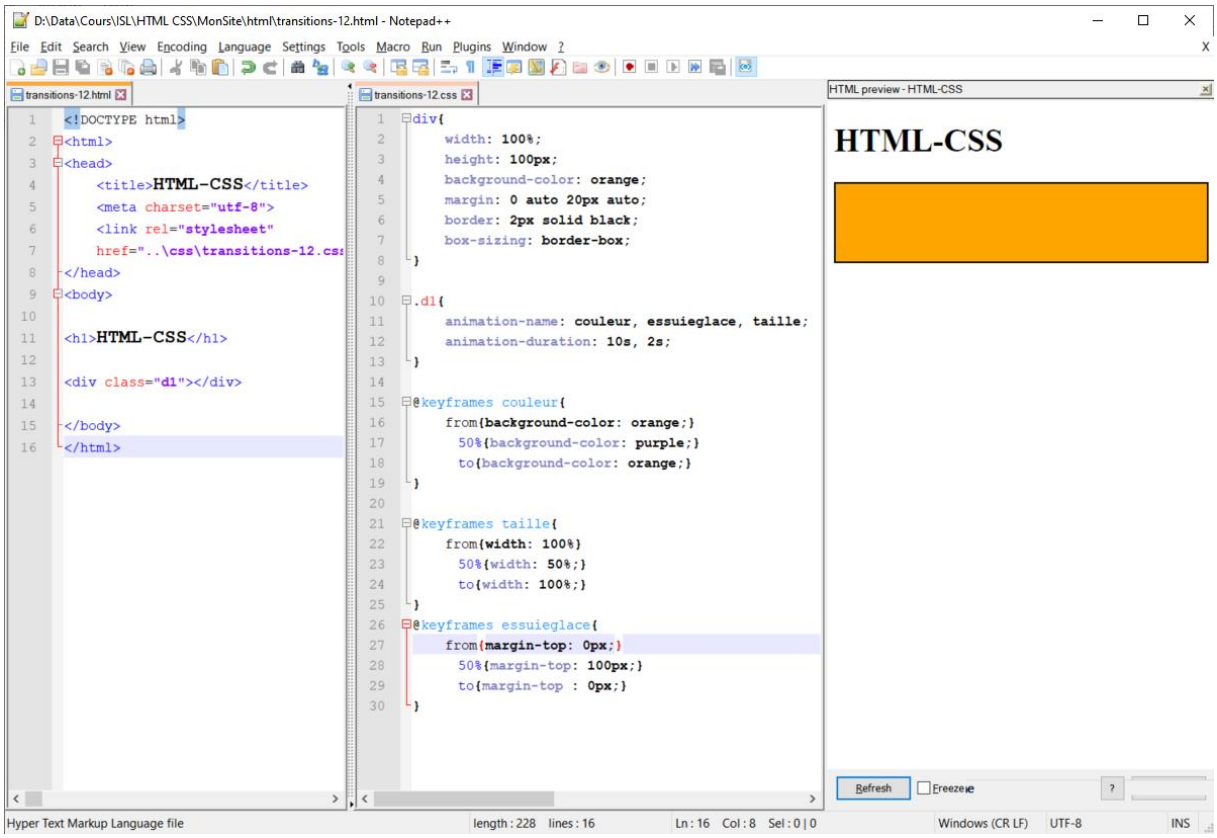
Maintenant que nous avons vu et compris le rôle de la règle @keyframes dans une animation, il est temps d'étudier les différentes propriétés animation-* qui vont nous permettre de gérer notre animation en soi avant de terminer avec la propriété raccourcie animation.

La propriété animation-name

La propriété animation-name va nous permettre de définir une liste d'animations qui doivent s'exécuter. On va donc lui passer un ou plusieurs noms qui devront correspondre aux noms des règles @keyframes qui définissent les propriétés à animer et les différentes valeurs qu'elle doit avoir pendant l'animation.

Si on fournit plusieurs noms à la propriété animation-name, alors il faudra définir à minima la durée de l'animation pour chaque animation avec la propriété animation-duration si on veut des animations fonctionnelles. Dans le cas où l'on fournit moins de valeurs à animation-duration qu'à animation-name, alors les animations supplémentaires vont réutiliser les valeurs de animation-duration dans l'ordre.

Par exemple, si on définit 5 animations avec animation-name et qu'on ne donne que 3 valeurs à animation-duration, alors la quatrième animation reprendra la première valeur de animation-duration et la cinquième animation reprendra sa deuxième valeur.



Dans l'exemple précédent, on définit 3 règles @keyframes qui définissent chacune des changements de valeurs pour une propriété. Dans le cas présent, on aurait aussi bien pu tout mettre dans une seule règle @keyframes. Cependant, il est généralement considéré comme une bonne pratique d'avoir une règle @keyframes pour une propriété puisque cela va nous permettre de nous resservir indépendamment d'une règle ou d'une autre pour l'appliquer à un élément ou à un autre.

Je passe ensuite les trois noms des animations que je souhaite exécuter à ma propriété animation-name. Ici, vous pouvez remarquer que je n'indique que deux valeurs de durée d'animation dans ma propriété animation-duration.

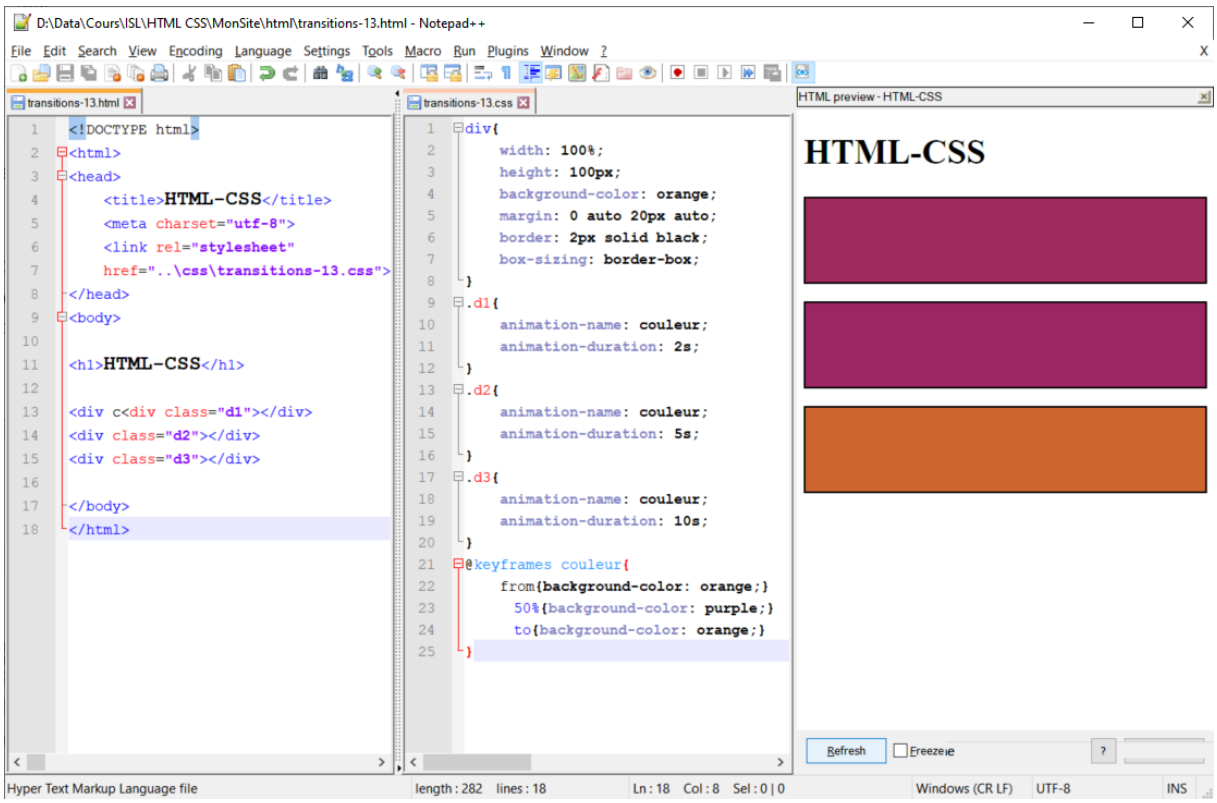
La première animation déclarée dans animation-name se servira de la première valeur, la deuxième de la deuxième valeur et la troisième à nouveau de la première valeur. L'animation « couleur » va

donc durer 10 secondes, l'animation « essuieglace » va durer 2 secondes et l'animation « taille » va durer 10 secondes.

Notez également ici que lorsqu'on définit plusieurs animations comme cela, les animations vont par défaut toutes se lancer en même temps et non pas à la suite les unes des autres.

La propriété animation-duration

La propriété animation-duration nous permet de définir le temps que doit durer une animation. On doit préciser une durée en secondes.




```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet"
    href="..\css\transitions-13.css">
</head>
<body>

<h1>HTML-CSS</h1>

<div c<div class="d1"></div>
<div class="d2"></div>
<div class="d3"></div>

</body>
</html>
```

```
div{
    width: 100%;
    height: 100px;
    background-color: orange;
    margin: 0 auto 20px auto;
    border: 2px solid black;
    box-sizing: border-box;
}
.d1{
    animation-name: couleur;
    animation-duration: 2s;
}
.d2{
    animation-name: couleur;
    animation-duration: 5s;
}
.d3{
    animation-name: couleur;
    animation-duration: 10s;
}
@keyframes couleur{
    from{background-color: orange;}
     50%{background-color: purple;}
     to{background-color: orange;}
}
```

Ici, on fournit la même règle @keyframes à chacun de nos trois div et on définit ensuite des paramètres d'animations différents à partir de cette règle en donnant des durées d'animation différentes. Une nouvelle fois, je vous rappelle qu'une règle @keyframes n'est qu'un cadre qui sert qu'à définir différentes valeurs pour une propriété à différents stades d'une animation quelconque.

Ensuite, n'importe quelle animation (qui va être construite et déclarée à proprement parler par les différentes propriétés animation-*) va pouvoir se resservir de cette règle @keyframes. C'est la raison pour laquelle je peux ici utiliser ma règle @keyframes dans trois animations pour trois éléments différents.

La propriété animation-timing-function

La propriété animation-timing-function va nous permettre comment doit progresser l'animation entre les différentes valeurs de keyframes : la progression de l'animation peut être linéaire, s'accélérer de plus en plus, etc.

Nous allons pouvoir passer les valeurs suivantes à animation-timing-function :

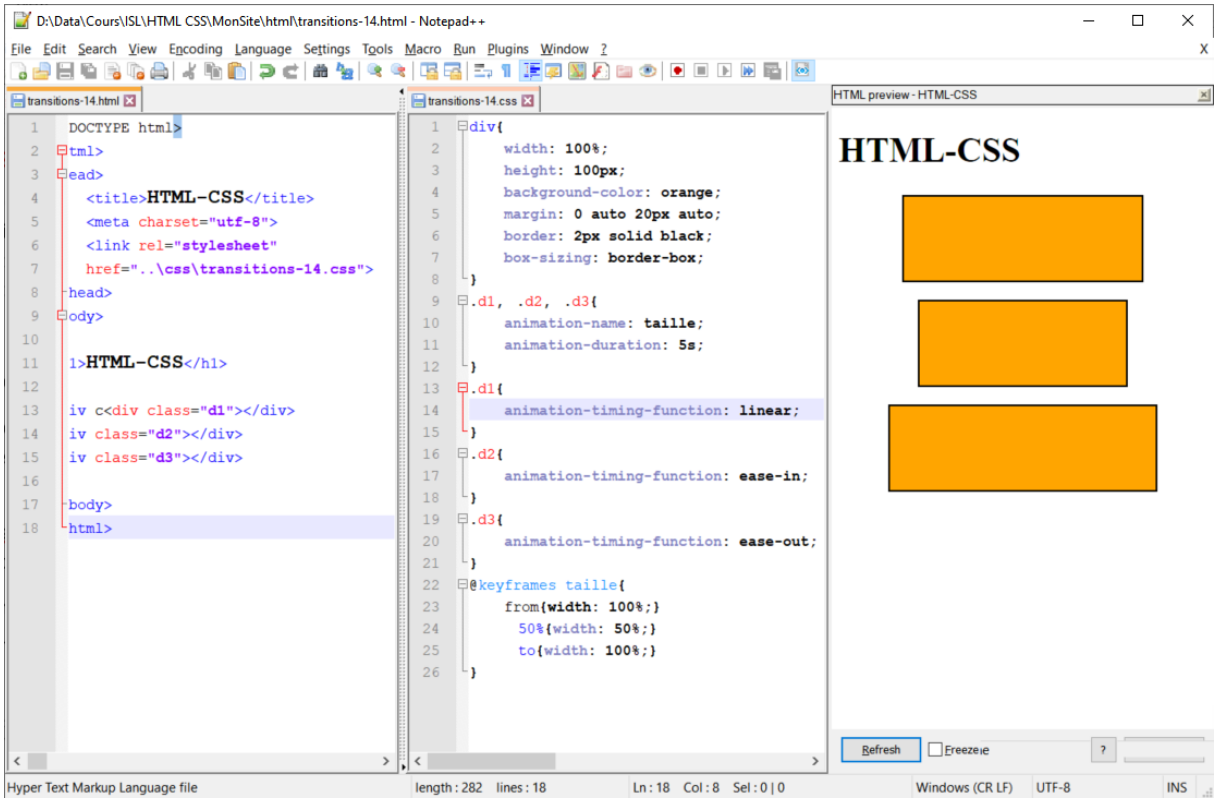
- ease : valeur par défaut. Entre deux valeurs de keyframes, l'animation va commencer relativement lentement, puis accélérer au milieu et se terminer lentement ;
- linear : Entre deux valeurs de keyframes, l'animation aura une vitesse constante du début à la fin ;
- ease-in : Entre deux valeurs de keyframes, l'animation va commencer lentement puis accélérer jusqu'à atteindre la prochaine valeur de keyframe ;
- ease-out : Entre deux valeurs de keyframes, l'animation va commencer rapidement et décélérer progressivement jusqu'à atteindre la prochaine valeur de keyframe ;
- ease-in-out : Entre deux valeurs de keyframes, l'animation commence lentement, accélère au milieu et finit lentement ;
- cubic-bezier(x1, y1, x2, y2) : permet de définir une courbe de Bézier spécifique pour créer une animation à la vitesse totalement contrôlée.

Notre but n'est ici bien évidemment pas de faire un cours sur les courbes de Bézier. Vous pouvez simplement retenir les équivalents mot-clef/Bézier suivants qui peuvent se révéler utiles :

- transition-timing-function : ease est équivalent à transition-timing-function : cubic-bezier(0.25, 0.1, 0.25, 1) ;
- transition-timing-function : ease-in est équivalent à transition-timing-function : cubic-bezier(0.42, 0, 1, 1) ;
- transition-timing-function : ease-out est équivalent à transition-timing-function : cubic-bezier(0, 0, 0.58, 1) ;
- transition-timing-function : ease-in-out est équivalent à transition-timing-function : cubic-bezier(0.42, 0, 0.58, 1).

Concernant la propriété animation-timing-function, vous devez bien comprendre que les valeurs passées vont dicter le comportement des animations entre chaque valeur de keyframes, c'est-à-dire que le pattern donné par animation-timing-function va être appliqué et répété entre chaque valeur de keyframes et ne va pas définir l'animation au complet.

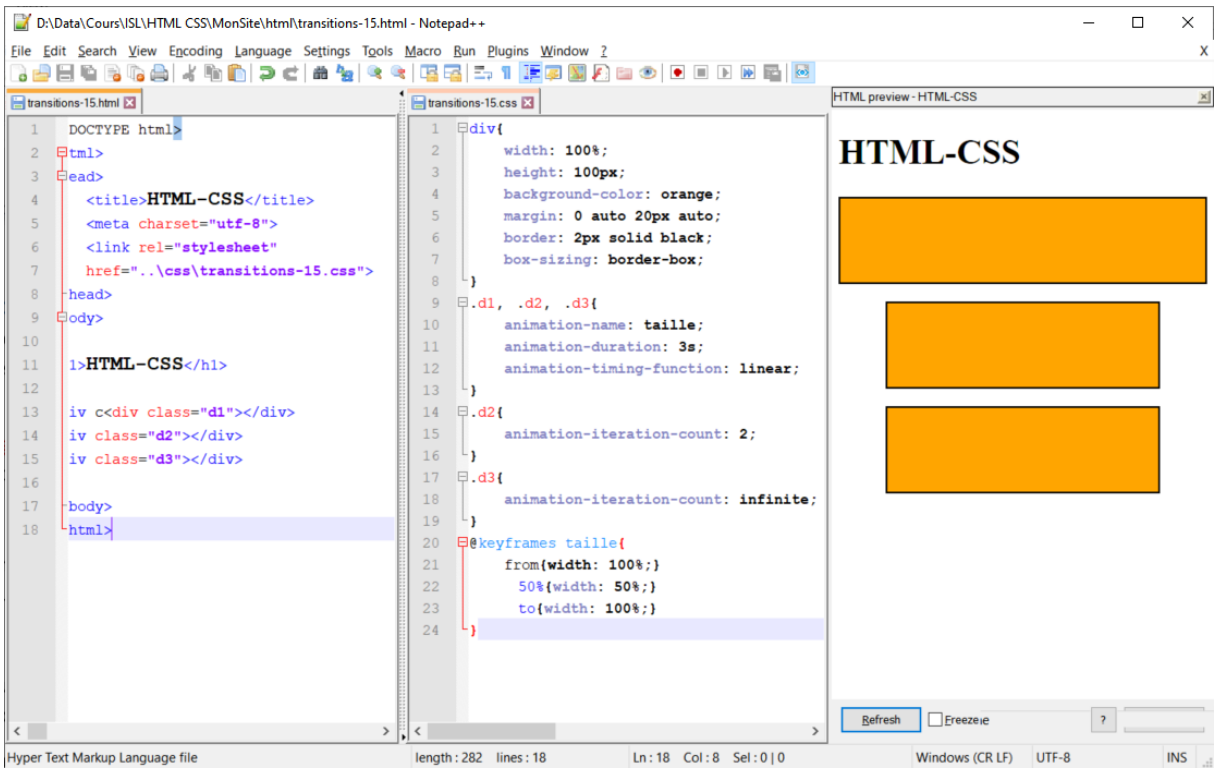
Par exemple, si on définit une règle @keyframes avec une valeur intermédiaire à 50% et que l'animation se fait selon animation-timing-function: ease, l'animation va commencer relativement lentement à partir du from, puis accélérer pour arriver lentement au 50% puis repartir relativement lentement à partir du 50% pour accélérer à nouveau et finir à nouveau lentement au niveau du to.



La propriété animation-iteration-count

La propriété animation-iteration-count va nous permettre de définir combien de fois une animation va être jouée. Par défaut, une animation ne sera jouée qu'une fois.

Pour modifier ce comportement par défaut, on va pouvoir passer soit un nombre à animation-iteration-count qui va correspondre au nombre de fois que l'on souhaite jouer l'animation, soit le mot clef infinite qui signifie que l'animation va se répéter à l'infini.



Ici, ma première animation (l'animation correspondante à mon div class="d1") ne va être jouée qu'une fois par défaut.

Ensuite, on demande à notre deuxième animation de se répéter, c'est-à-dire d'être jouée deux fois avec animation-iteration-count: 2.

Finalement, notre troisième animation va se répéter à l'infini grâce à animation-iteration-count: infinite.

La propriété animation-direction

La propriété animation-direction va nous permettre de spécifier le sens dans lequel une animation doit être jouée, c'est-à-dire si elle doit être jouée en partant du début ou de la fin pour une ou plusieurs de ses itérations ou répétitions.

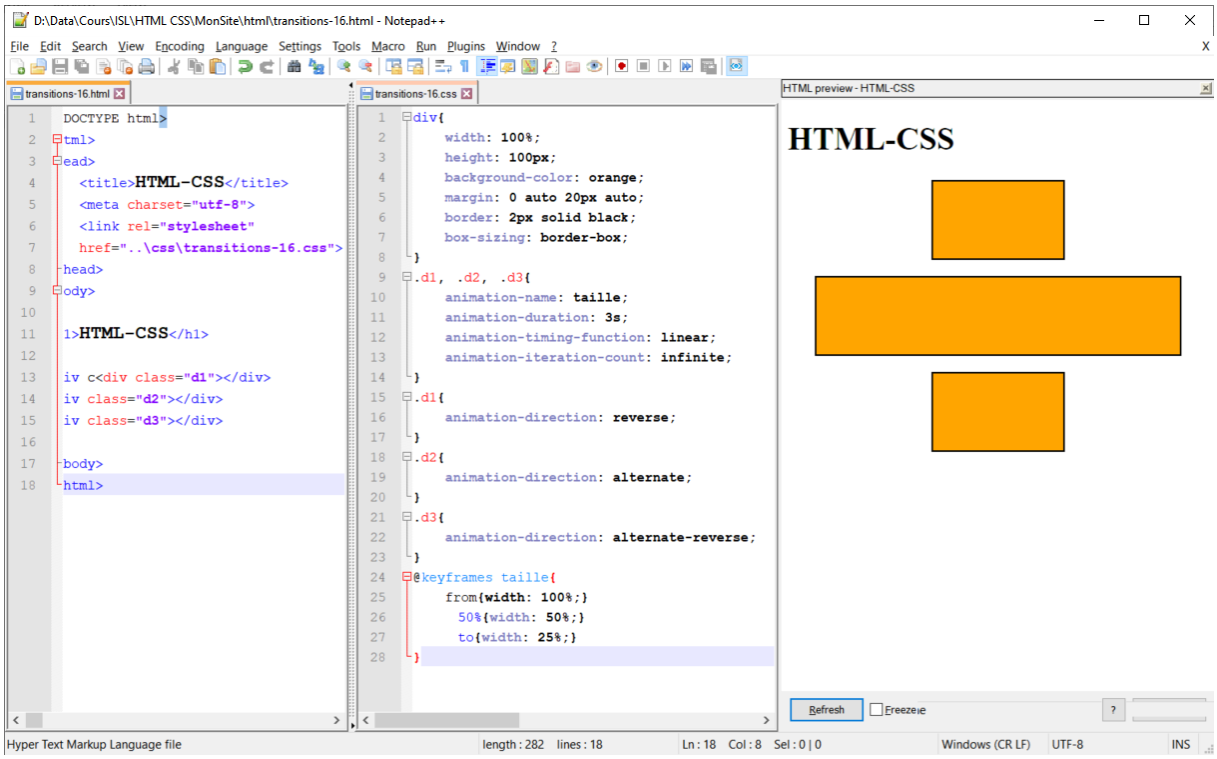
Nous allons pouvoir passer les valeurs suivantes à animation-direction :

- normal : valeur par défaut. L'animation est jouée dans le sens dans lequel elle a été déclarée (du from vers le to) ;
- reverse : l'animation est jouée dans le sens inverse pour toutes ses itérations ;
- alternate : l'animation va être jouée une première fois dans le sens normal, puis dans le sens contraire, puis à nouveau dans le sens normal et etc. ;

- alternate-reverse : l'animation va être jouée une première fois dans le sens inverse, puis dans le sens normal, puis à nouveau dans le sens inverse et etc..

Notez qu'en inversant le sens d'une animation pour une ou plusieurs de ses itérations, les propriétés liées au timing seront également inversées. Par exemple, une animation possédant une animation-direction : reverse et une animation-timing-function : ease-in sera jouée comme si la valeur était animation-timing-function : ease-out.

Notez également que les valeurs animation-direction : alternate et animation-direction : alternate-reverse ne vont évidemment n'avoir un effet que pour les animations qui vont être jouées plus d'une fois.



Ici, nous créons une règle @keyframes qui va modifier la valeur de la propriété width de 100% vers 25%. Dans son sens normal, l'animation fait donc passer nos div d'une taille de 100% à 25%.

Ensuite, nous demandons à chacune de nos animations de se répéter à l'infini pour bien voir le comportement lié aux valeurs alternate et alternate-reverse.

Pour notre première animation, cependant, on indique une propriété animation-direction : reverse qui signifie qu'on souhaite que l'animation joue dans le sens contraire. Le départ de notre animation va donc utiliser la taille width: 25% et la taille du div va grandir jusqu'à 100%.

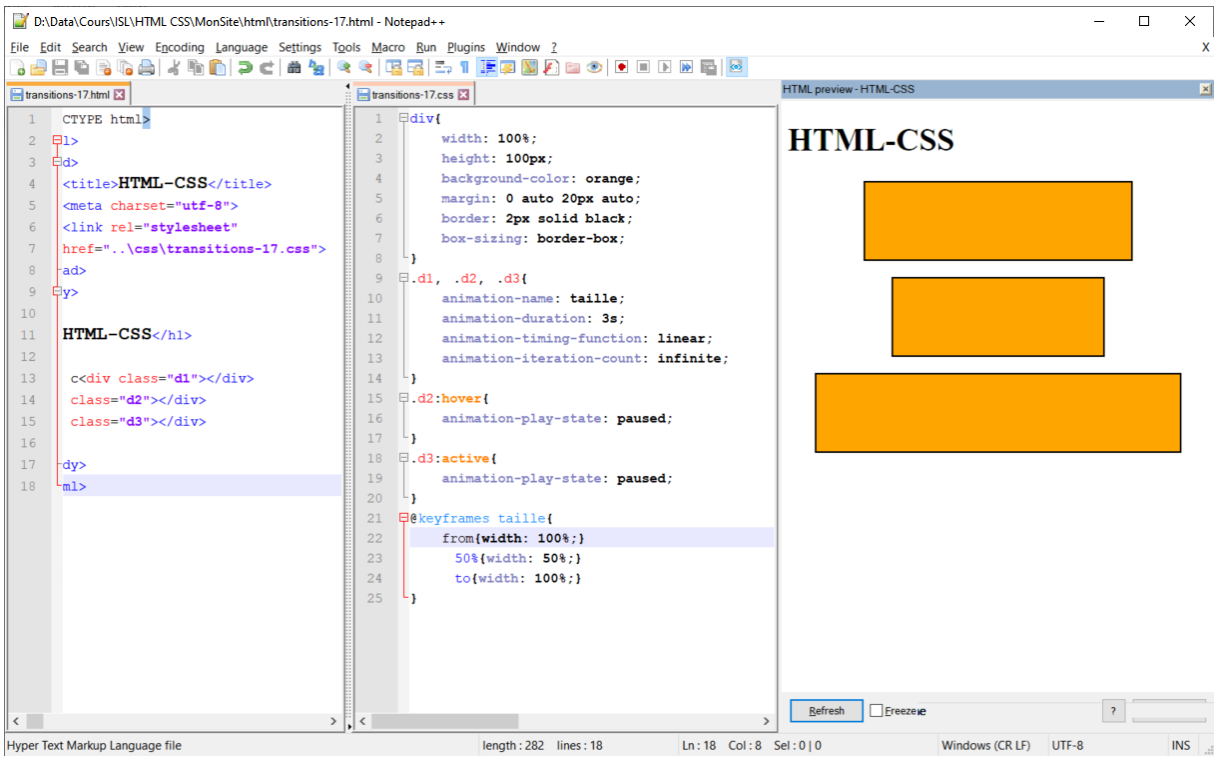
Pour notre deuxième animation, on passe la valeur alternate à la propriété animation-direction. Notre animation va donc jouer une première fois dans le sens normal, puis dans le sens contraire, puis à nouveau dans le sens normal et etc.

La dernière animation va avoir le comportement opposé de la deuxième : elle va commencer à jouer dans le sens inverse puis alterner à chaque nouvelle itération (ou répétition) puisqu'on lui a donné une animation-direction : alternate-reverse

La propriété animation-play-state

La propriété animation-play-state va nous permettre de définir si une animation doit être jouée ou être en pause. On va pouvoir lui passer la valeur running (l'animation s'exécute normalement) ou paused (l'animation est mise en pause).

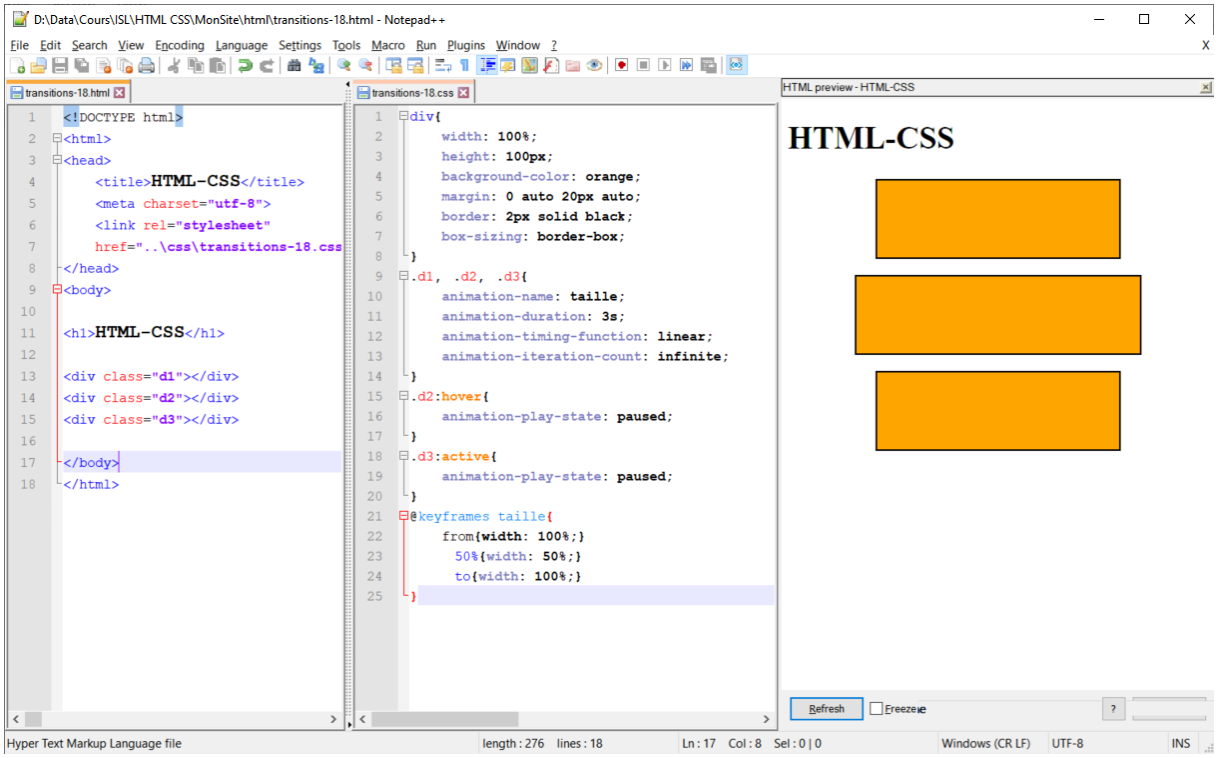
Cette propriété va pouvoir être utile pour mettre une animation en pause à un certain point de l'animation ou selon une certaine action de l'utilisateur. Par exemple, on va pouvoir proposer aux utilisateurs de mettre en pause une animation lorsqu'ils passant le curseur de leur souris sur l'élément pour lequel une animation est jouée ou lorsqu'ils cliquent (en gardant le clic enfoncé) sur l'élément en utilisant les pseudo classes :hover et :active.



La propriété animation-delay

La propriété animation-delay va nous permettre de définir quand une animation doit commencer c'est-à-dire s'il doit y avoir un délai avant le lancement de l'animation.

On va devoir passer une valeur en secondes à animation-delay qui va correspondre au délai qu'il doit s'écouler avant le lancement de l'animation.



La propriété animation-fill-mode

Par défaut, une règle @keyframes ou plus généralement une animation ne va pas affecter ni donc définir la valeur de la propriété animée en dehors du temps de l'animation.

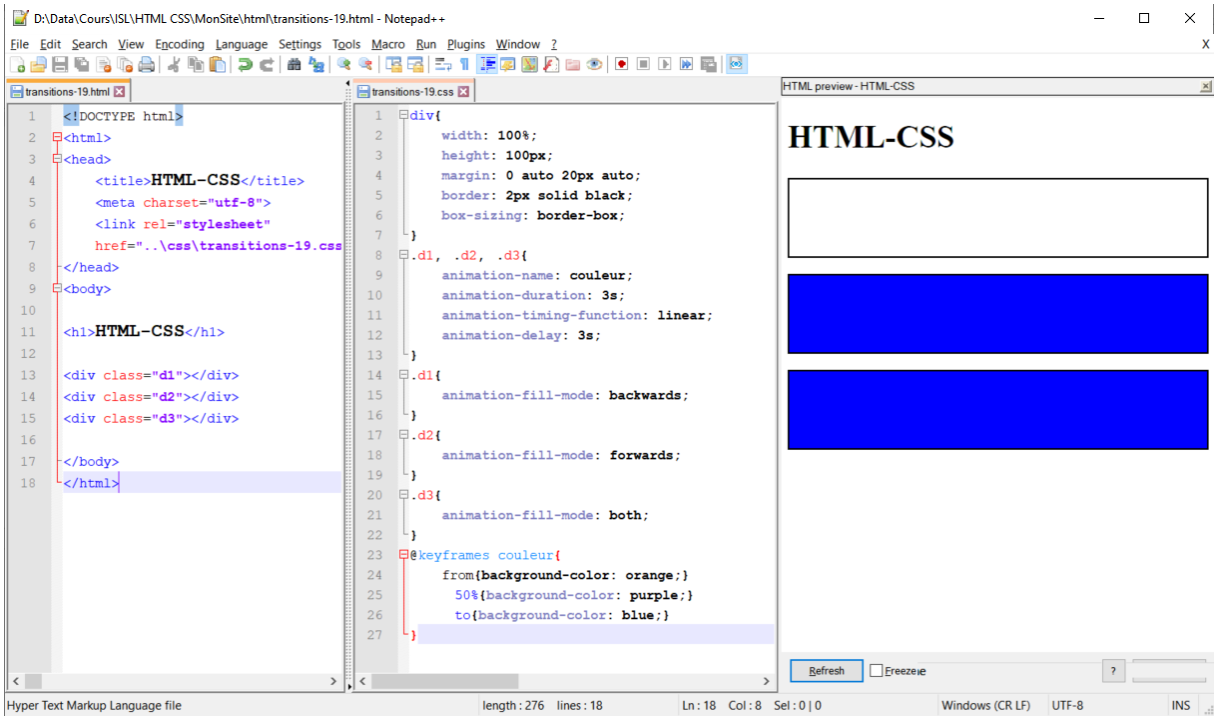
Nous avons vu cela en début de tutoriel et c'est la raison pour laquelle je définis depuis le début de celui-ci un comportement « normal » ou par défaut pour les propriétés animées en CSS.

Cependant, il existe un moyen plus élégant de faire cela en utilisant la propriété animation-fill-mode. Cette propriété va en effet nous permettre conserver le comportement de nos propriétés défini dans l'animation en dehors de l'animation (avant ou après).

On va ainsi pouvoir appliquer la valeur de départ de notre animation à notre propriété avant que l'animation ne commence ou la valeur d'arrivée de notre animation à la propriété après que celle-ci soit finie.

Pour faire cela, nous allons pouvoir passer l'une des valeurs suivantes à animation-fill-mode :

- backwards : notre propriété animée utilisera la valeur de départ de l'animation comme valeur avant que l'animation ne commence ;
- forwards : notre propriété animée utilisera la valeur de fin de l'animation comme valeur après que l'animation soit terminée ;
- both : notre propriété animée utilisera la valeur de départ de l'animation comme valeur avant que l'animation ne commence et la valeur de fin de l'animation comme valeur après que l'animation soit terminée.



Dans l'exemple ci-dessus, j'applique un délai de 3 secondes à chacune de mes animations.

La première animation a un animation-fill-mode : backwards, elle utilisera donc la valeur de départ de l'animation durant le délai tout comme la dernière animation qui possède un animation-fill-mode : both.

Pour la deuxième animation, au contraire, la propriété n'utilisera pas de valeur avant le début de l'animation puisque rien ne lui a été précisé.

Après la fin de l'animation, c'est l'inverse qui va se passer : notre deuxième animation va conserver la valeur de fin de l'animation de la propriété grâce à animation-fill-mode : forwards tandis que notre première animation n'aura plus aucune valeur attachée à son background-color.

La propriété animation

La propriété animation correspond à la version raccourcie ou notation short-hand des propriétés animation-* vues ci-dessus.

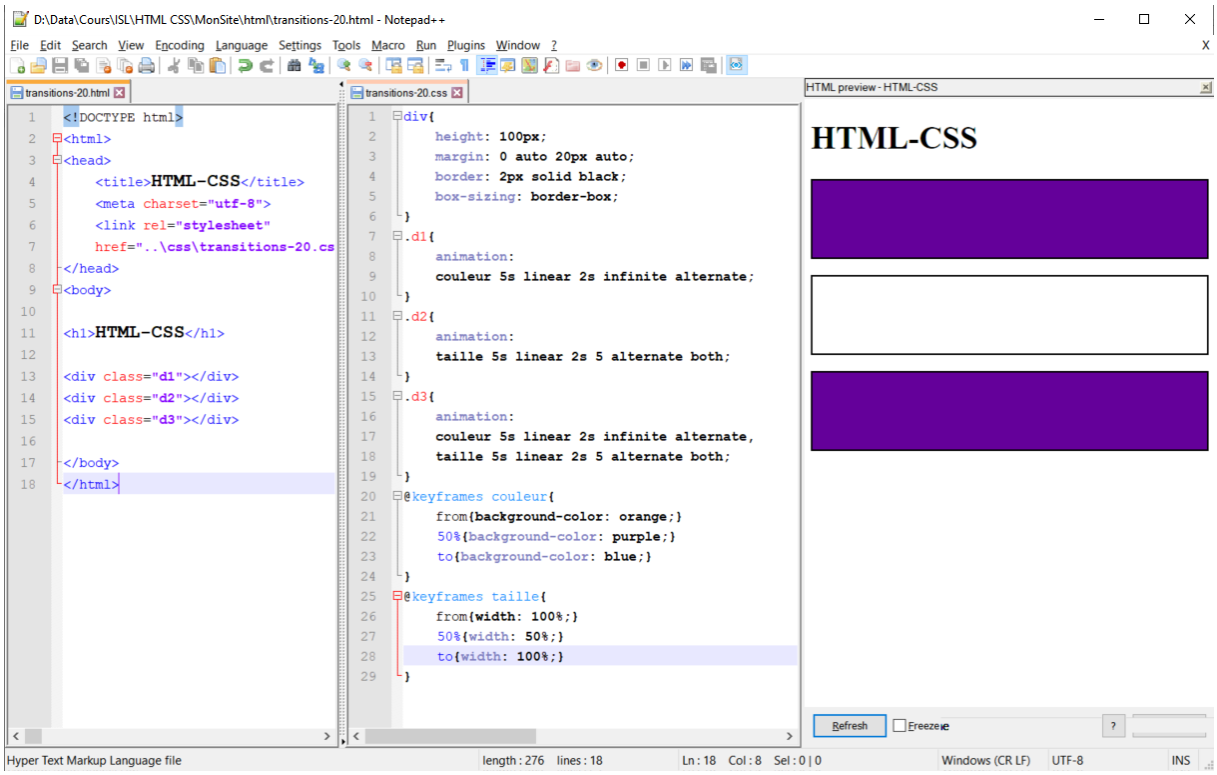
Nous allons pouvoir lui passer les différentes valeurs relatives aux propriétés animation- pour créer simplement des animations complètes. Il est généralement considéré comme une bonne pratique de passer les valeurs dans l'ordre suivant pour être certain que l'animation fonctionne correctement :

1. La valeur relative à la propriété animation-name ;
2. La valeur relative à la propriété animation-duration ;
3. La valeur relative à la propriété animation-timing-function ;

4. La valeur relative à la propriété animation-delay ;
5. La valeur relative à la propriété animation-iteration-count ;
6. La valeur relative à la propriété animation-direction ;
7. La valeur relative à la propriété animation-fill-mode ;
8. La valeur relative à la propriété animation-play-state .

Bien évidemment, nous n'allons pas être obligé de préciser toutes les valeurs pour chaque animation : si une valeur est omise, la valeur par défaut de la propriété correspondante sera utilisée. Cependant, notez bien que la première valeur de type « secondes » fournie à animation sera toujours considérée comme étant la durée de l'animation tandis que la deuxième sera toujours considérée comme le délai.

On va également pouvoir créer plusieurs animations avec la propriété animation. Pour cela, il suffira de séparer les déclarations des différentes animations par une virgule.



Ici, dans notre premier exemple, on anime la couleur de fond de notre élément div sur une durée de 5 secondes. L'animation doit progresser de façon linéaire et commencer après un délai de 2 secondes. De plus, on demande à l'animation de se répéter à l'infini et d'alterner le sens dans lequel elle est jouée à chaque fois.

Dans notre deuxième exemple, on anime la taille du div sur 5 secondes. L'animation doit à nouveau progresser de façon linéaire et commencer après un délai de 2 secondes. L'animation doit se répéter

5 fois et nous demandons à ce que la propriété utilise la valeur de départ de l'animation avant que celle-ci ne commence et la valeur d'arrivée après la fin de l'animation.

Notre troisième exemple fait jouer les deux animations précédentes en même temps. Pour cela, on sépare les différentes déclarations par une virgule dans animation, tout simplement.

Créer des transformations en CSS

Dans les leçons précédentes, nous avons pu étudier les transitions et les animations en CSS qui permettent de modifier la valeur de certaines propriétés progressivement en fonction de certains critères (chargement de la page, passage de souris sur l'élément, etc.) et donc d'ajouter un côté interactif à nos pages web.

Le CSS va également nous permettre d'appliquer des transformations à nos éléments : on va pouvoir incliner nos éléments, les déformer, les translater, etc.

Dans cette leçon, nous allons expliquer en détail comment fonctionnent les transformations et apprendre à créer des transformations plus ou moins complexes.

Définir une transformation en CSS

La possibilité d'effectuer des transformations en CSS est récente et les possibilités et fonctionnalités des transformations sont donc aujourd'hui relativement limitées.

Cependant, on peut s'attendre à ce que de nouvelles fonctionnalités viennent s'ajouter aux transformations dans un futur proche.

A terme, les transformations en CSS devraient être définies par 3 critères qui vont pouvoir être renseignés via 3 propriétés CSS différentes :

- La propriété transform-box va nous permettre de définir une boite de référence qui va être utilisée pour calculer le point d'origine et pour réaliser la transformation en soi ;
- La propriété transform-origin va nous permettre de définir un point d'origine à partir duquel réaliser la transformation ;
- La propriété transform va nous permettre de définir un effet de transformation.

La propriété transform-box ne fait pas encore partie des recommandations du W3C et est toujours en développement. Il est donc déconseillé de l'utiliser pour l'instant puisque sa définition n'est pas encore stable et que le support par les navigateurs n'est pas assuré. Nous ne l'étudierons ainsi pas ici.

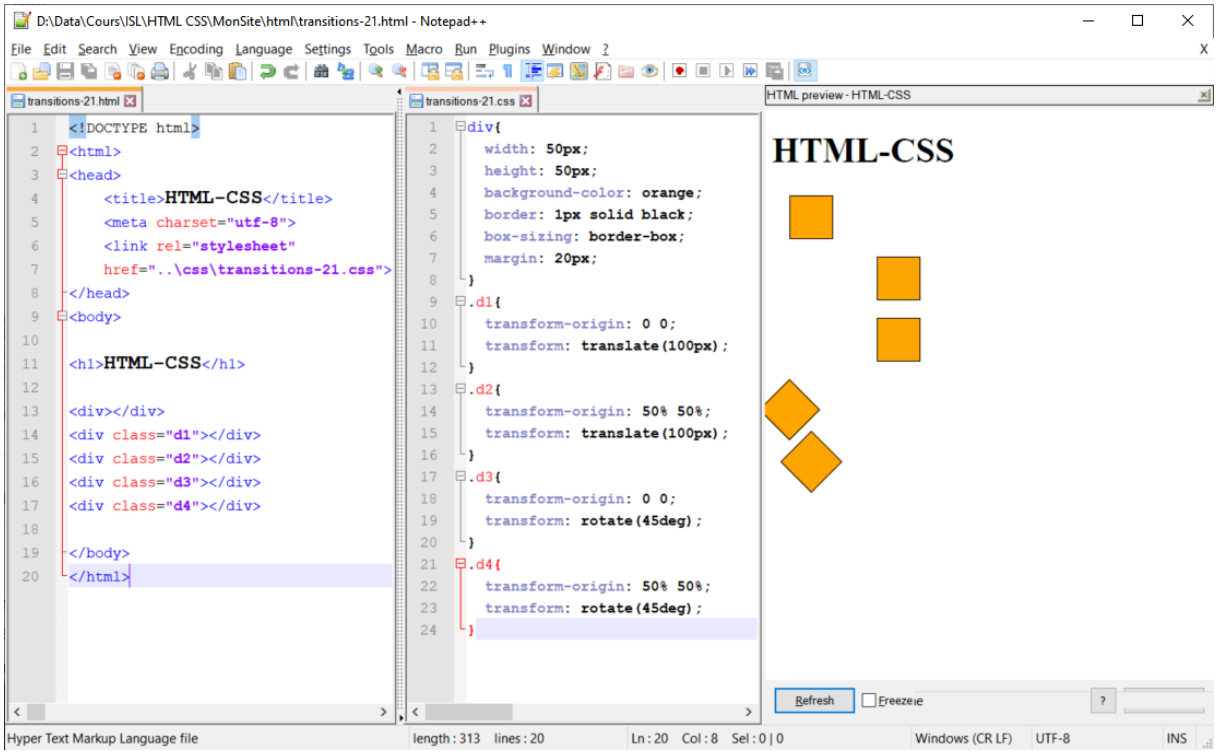
La propriété transform-origin permet de définir un point d'origine pour une transformation. Par défaut, le point d'origine est le centre de l'élément.

Cette propriété va pouvoir prendre une ou deux valeurs. En n'indiquant qu'une valeur, la valeur sera utilisée pour définir les coordonnées dans l'axe horizontal et dans l'axe vertical du point d'origine. En indiquant deux valeurs, la première valeur va permettre de définir la coordonnée horizontale du point d'origine tandis que la seconde valeur va permettre de définir sa coordonnée verticale.

Nous allons pouvoir passer les mots clefs top, right, bottom, left et center ainsi que des longueurs ou des pourcentages à cette propriété. Les valeurs de type longueur ou pourcentage vont définir l'éloignement du point d'origine à partir du bord supérieur gauche de la boite de référence.

La propriété transform va nous permettre de définir un ou plusieurs effets de transformation à appliquer à un élément : inclinaison, rotation, déformation, etc.

Cette propriété va accepter différents mots clefs ou plus exactement différentes fonctions qui vont définir le type de transformation qui va être appliqué à un élément. Nous allons étudier ces valeurs dans la suite de la leçon.




```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet"
    href="..\css\transitions-21.css">
</head>
<body>

<h1>HTML-CSS</h1>

<div></div>
<div class="d1"></div>
<div class="d2"></div>
<div class="d3"></div>
<div class="d4"></div>

</body>
</html>
```

```
div{
    width: 50px;
    height: 50px;
    background-color: orange;
    border: 1px solid black;
    box-sizing: border-box;
    margin: 20px;
}
.d1{
    transform-origin: 0 0;
    transform: translate(100px);
}
.d2{
    transform-origin: 50% 50%;
    transform: translate(100px);
}
.d3{
    transform-origin: 0 0;
    transform: rotate(45deg);
}
.d4{
    transform-origin: 50% 50%;
    transform: rotate(45deg);
}
```

Dans le code ci-dessus, nous avons défini 4 transformations pour nos 4 div portant des attributs class.

Nous appliquons un premier effet de transformation à nos deux premiers div qui va être une translation, c'est-à-dire un déplacement selon une direction. Ici, on va donc déplacer nos deux div de 100px à partir de leur point d'origine vers la droite.

Ensuite, nous appliquons un effet de rotation à nos deux derniers div. La rotation va s'effectuer dans le sens des aiguilles d'une montre.

Le point d'origine de la transformation pour le premier div est son coin supérieur gauche tandis que cela va être le centre pour le deuxième. Comme la transformation n'est ici qu'un déplacement horizontal, modifier le point d'origine ne change pas le résultat de la transformation.

En revanche, cela va être différent pour une rotation : notre troisième div va pivoter autour d'un point central qui va être son coin supérieur gauche tandis que notre quatrième div va pivoter autour de son point central. Dans ce cas-là, modifier le point d'origine de la transformation change le résultat obtenu.

Finalement, vous pouvez noter ici qu'à la différence des transitions et des animations pour lesquelles on précise une durée, les transformations sont permanentes et les éléments vont conserver la transformation.

Exemples de transformations 2D

Nous allons pour le moment nous concentrer sur les effets de transformations en deux dimensions. En effet, vous devez savoir qu'on peut également aujourd'hui créer des transformations en 3D en rajoutant une perspective à nos éléments. Nous discuterons de cette possibilité plus tard.

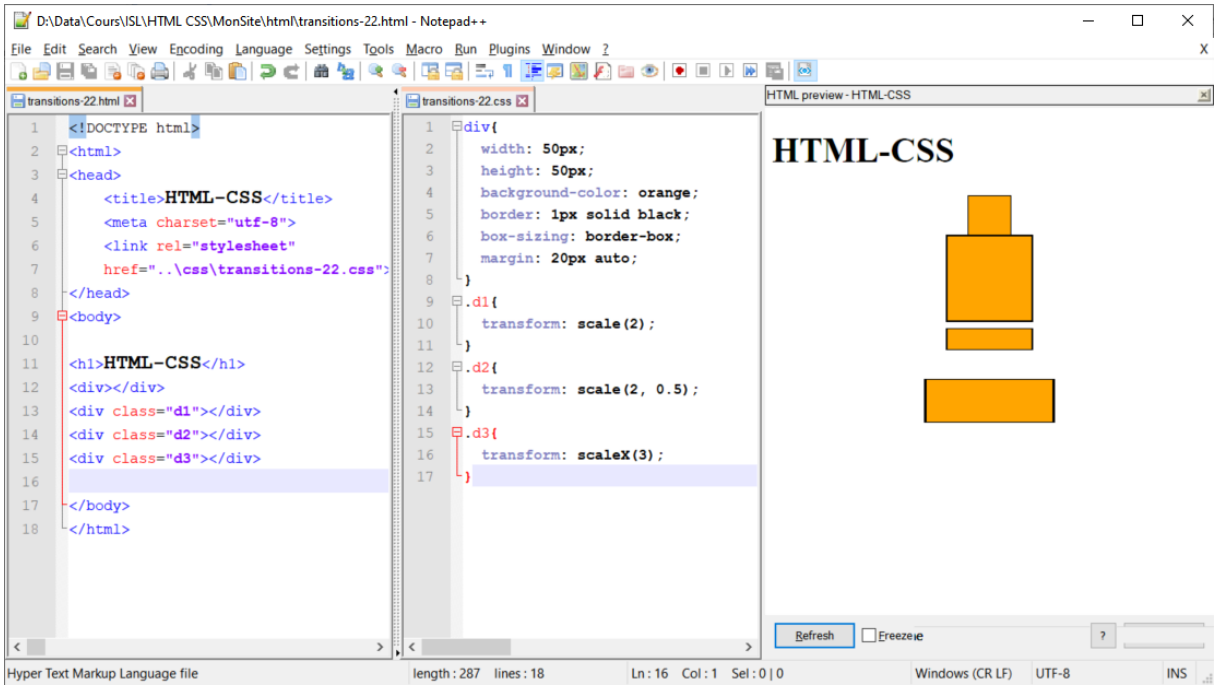
Modifier la taille ou l'échelle d'un élément avec scale()

La fonction scale() permet de modifier la taille ou plus exactement l'échelle de l'élément. Nous allons pouvoir lui passer deux nombres qui vont correspondre au pourcentage d'agrandissement en largeur et en hauteur de l'élément.

Par exemple, en écrivant transform : scale(2, 0.5), l'élément va doubler en largeur et être diminué de moitié en hauteur.

Notez que cette fonction ne va pas affecter que les propriétés width et height de l'élément mais également s'appliquer au font-size et au padding.

Finalement, vous devez savoir que la fonction scale() est la notation raccourcie des fonctions scaleX() et scaleY() qui permettent de ne modifier que la largeur et la hauteur respectivement d'un élément.



Ici, on ne passe qu'une valeur à notre fonction scale() pour notre première transformation : cette valeur va donc être utilisée pour calculer la mise à l'échelle horizontale et verticale et notre div fera deux fois sa taille d'origine.

Pour notre deuxième transformation, on demande à ce que le div devienne deux fois plus large et qu'il soit deux fois moins haut.

Finalement, on utilise la fonction scaleX() pour définir notre dernière transformation qui ne va donc impacter que la largeur de l'élément.

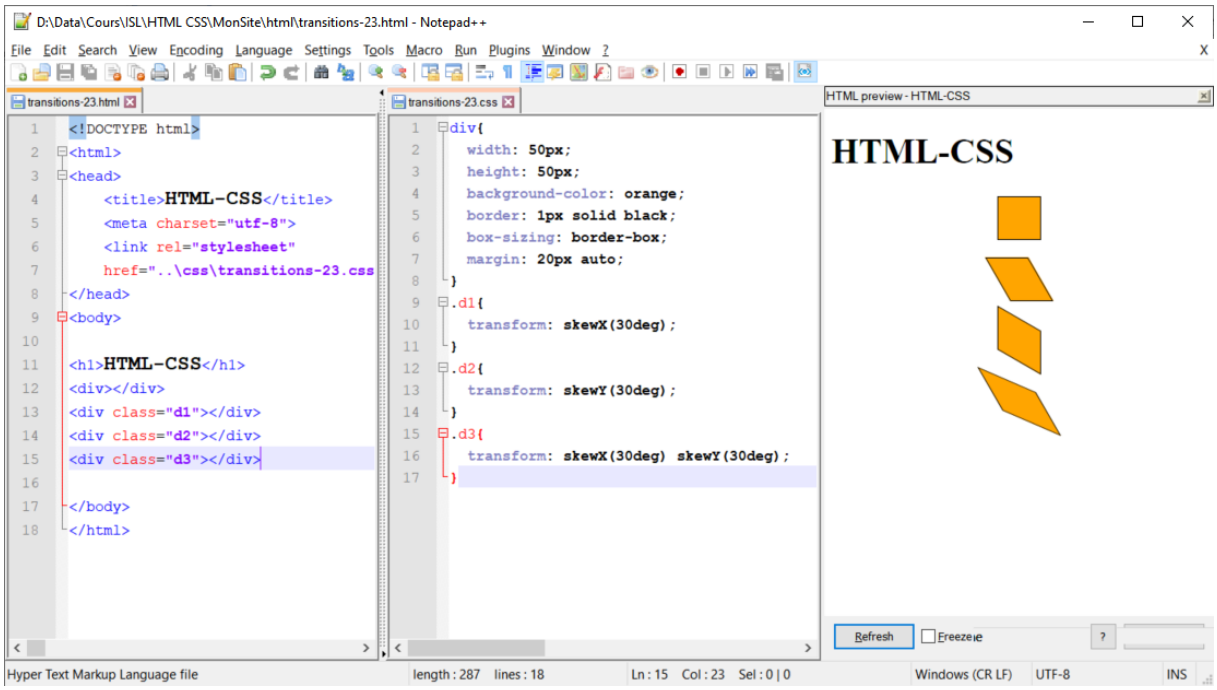
Notez bien ici qu'appliquer une transformation à des éléments ne va pas modifier la taille de l'espace qui leur était attribué à la base ni donc faire bouger les autres éléments autour en conséquence. Il faudra donc faire bien attention à ce que nos éléments transformés chevauchent pas leurs voisins et ne dépassent pas de leur parent.

Déformer un élément avec skewX() et skewY()

Avant toute chose, il faut savoir que la fonction skew(), si elle existe, n'a été créée que pour des questions de compatibilité et qu'on ne devrait jamais l'utiliser.

A la place, il faudra plutôt utiliser les fonctions skewX() et skewY() qui vont nous permettre de déformer un élément selon son axe horizontal ou vertical.

Nous allons devoir passer un angle (généralement en deg) à ces deux fonctions qui va représenter l'angle selon lequel l'élément doit être déformé le long de l'axe correspondant.

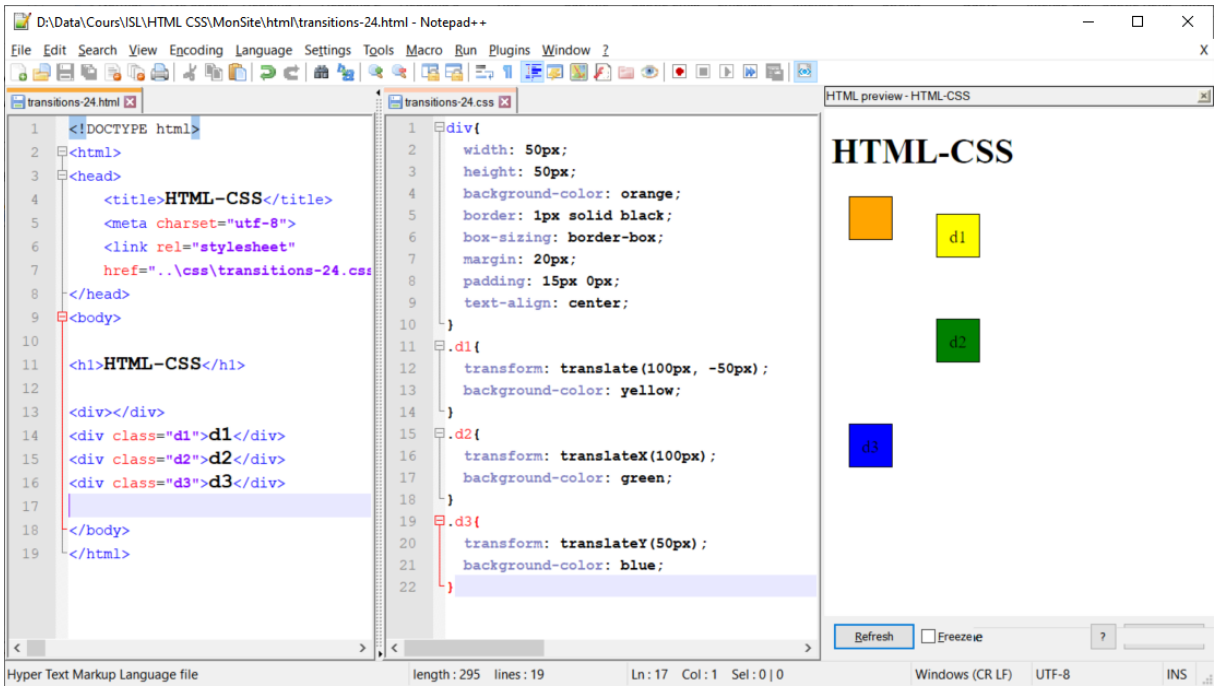


Effectuer une translation avec translate(X,Y)

La fonction translate() va nous permettre de créer une translation, c'est-à-dire de déplacer un élément selon un certain vecteur (ou selon une certaine distance et direction, pour faire simple).

Là encore, on va pouvoir passer deux valeurs de type longueur à translate() pour spécifier les caractéristiques de la translation dans l'axe horizontal et dans l'axe vertical ou utiliser les versions complètes translateX() et translateY().

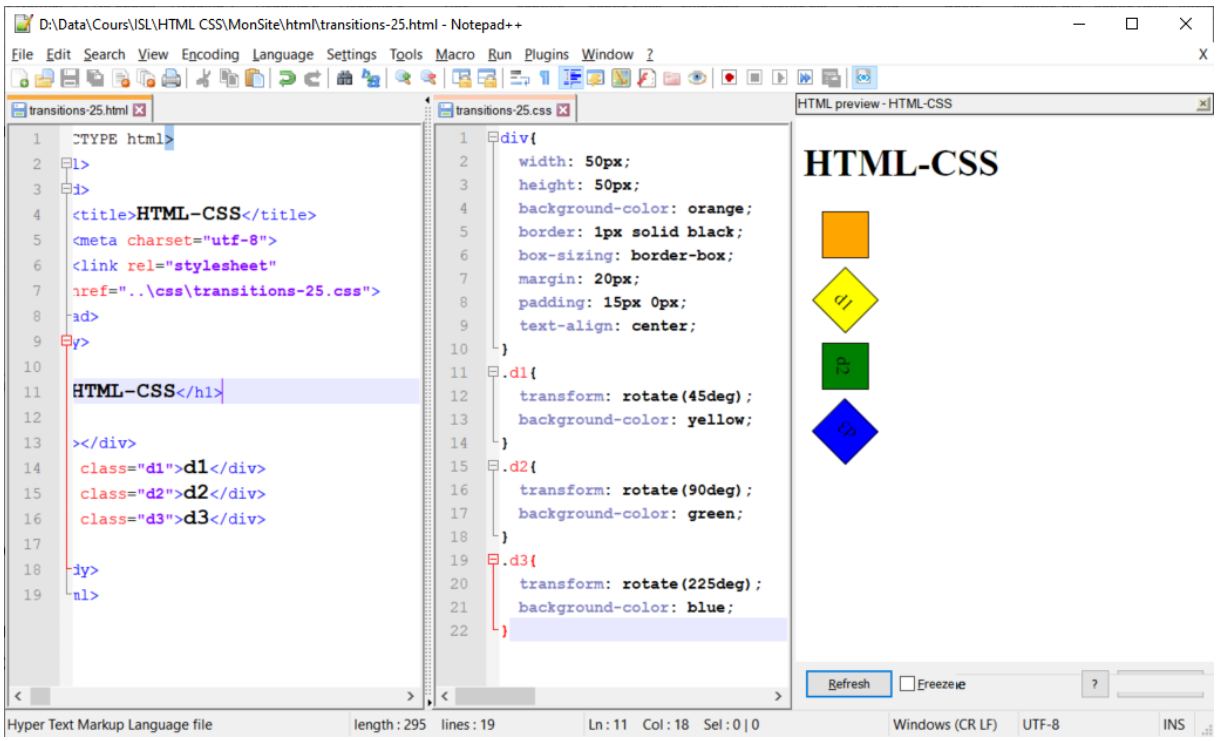
Les valeurs passées vont pouvoir être positives ou négatives. Une valeur positive pour l'axe horizontal va déplacer l'élément vers la droite tandis qu'une valeur positive pour l'axe vertical va déplacer l'élément vers le bas et inversement pour des valeurs négatives.



Effectuer une rotation avec rotate()

La fonction rotate() va nous permettre de faire pivoter un élément ou de lui faire effectuer une rotation selon un certain angle. Nous allons pouvoir lui fournir un angle (généralement en deg) en valeur.

La rotation va s'effectuer dans le sens des aiguilles d'une montre. Ainsi, indiquer rotate(90deg) va faire pivoter l'élément d'un quart de tour.



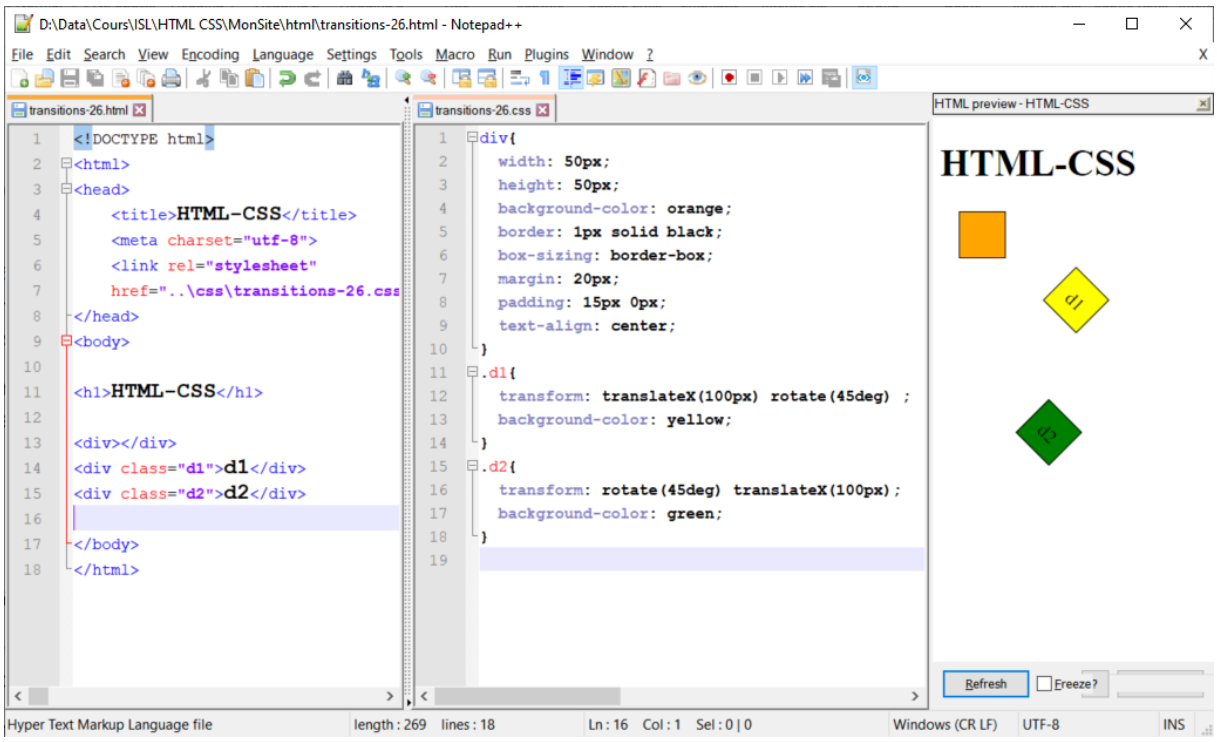
Définir une matrice de transformation avec matrix()

La fonction matrix() va nous permettre de définir notre propre matrice de transformation. Son utilisation est réservée aux personnes qui possèdent un bon degré de connaissance des transformations ET en mathématiques. Comme son usage est très marginal, je ne pense pas qu'il soit pertinent d'expliquer son fonctionnement en détail dans ce cours.

Appliquer plusieurs transformations d'un coup

Nous allons pouvoir définir plusieurs transformations à appliquer à un élément avec transform. Pour cela, il va suffire d'indiquer les différents effets de transformation à la suite.

Notez bien ici que les transformations ne vont pas toutes s'effectuer en même temps mais plutôt les unes à la suite des autres selon leur ordre de déclaration.



Expliquons le code ci-dessus. Ici, on applique les mêmes transformations à nos deux div mais pas dans le même ordre.

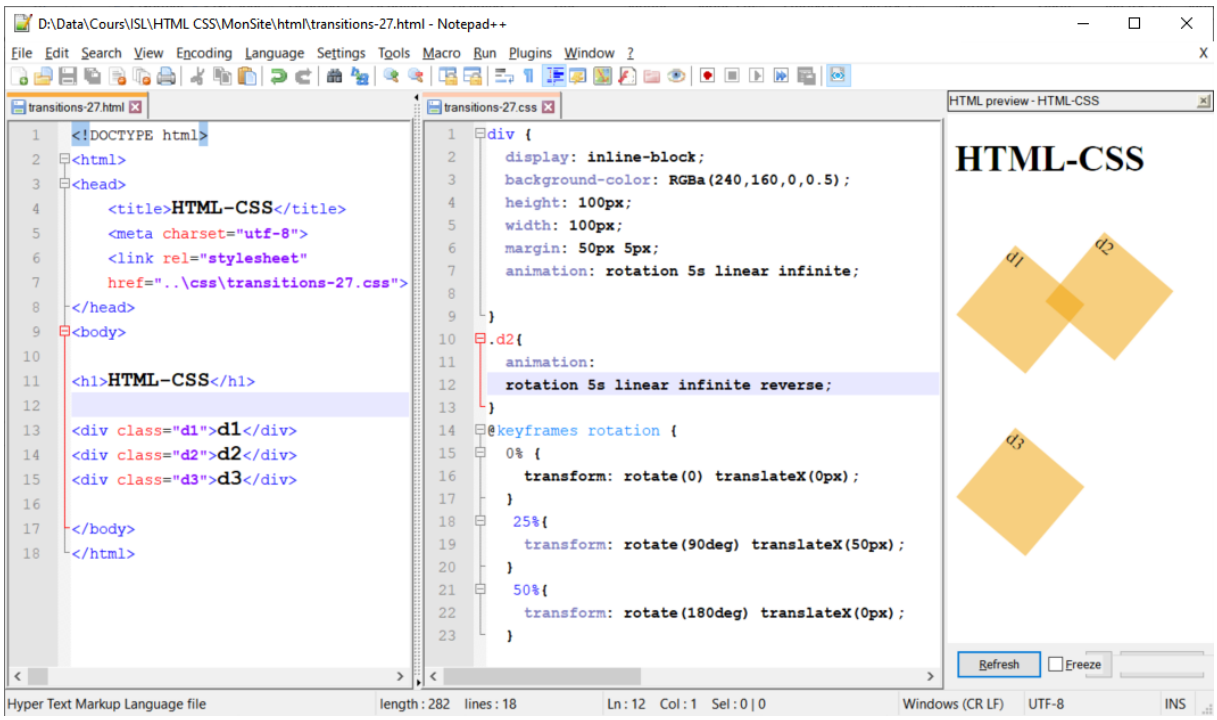
Le premier div auquel on applique une transformation va d'abord effectuer une translation de 100px vers la droite puis une rotation de 45 degrés dans le sens horaire.

Le deuxième div, au contraire, va lui d'abord effectuer une rotation de 45 degrés puis une translation de 100px. La différence est ici qu'après sa rotation l'axe horizontal du div (qu'il serait plus correct d'appeler l'axe des abscisses ou axe des X) est également incliné de 45 degrés et la translation va se faire selon cet axe.

Notre second div va donc être décalé de 100px selon cet axe incliné à 45 degrés.

Animer des transformations

Notez enfin qu'on va tout à fait pouvoir utiliser les transformations au sein d'animations en CSS. Cela va nous permettre d'ajouter un certain dynamisme à nos pages web.



Les transformations 3D

Finalement, vous devez savoir qu'on va également pouvoir créer des transformations en 3D, c'est-à-dire en rajoutant un axe Z qui va créer une perspective. Cet axe va nous permettre de simuler une profondeur et on va pouvoir ainsi faire comme si des éléments se rapprochaient ou s'éloignaient de l'utilisateur.

Avec les transformations 3D, nous commençons à toucher à des choses vraiment complexes en CSS et qui ne sont pas forcément pertinentes dans le cadre de ce cours complet car elles sont très peu utilisées et très spécifiques.

J'aborde ce sujet par souci d'exhaustivité mais je vais me contenter de vous expliquer rapidement le principe de fonctionnement des transformations 3D et vous donner quelques exemples qui me semblent les plus pertinents car encore une fois je ne pense pas qu'il soit pertinent de s'arrêter trop longtemps sur ces notions et d'essayer à tout prix de comprendre comment elles fonctionnent en détail.

Pour qu'une transformation 3D ait l'effet visuel escompté, il va déjà falloir créer un effet de perspective c'est-à-dire créer une impression d'espace 3D. Pour cela, nous allons utiliser la fonction perspective() avec nos effets de transformation.

La valeur passée à la fonction perspective() va définir l'intensité de l'effet 3D. On peut considérer qu'elle va représenter la distance entre l'utilisateur et l'élément. Plus la valeur passée à perspective() va être grande, plus l'élément sera éloigné de l'utilisateur au départ et moins l'effet 3D sera visible.

Ensuite, nous allons pouvoir définir nos effets de transformation 3D : mises à l'échelle en 3D, translations 3D ou rotations 3D. Nous allons également pouvoir créer nos propres transformations 3D grâce à la fonction matrix3d() mais il faudra pour cela comprendre comment fonctionne le calcul matriciel en mathématiques.

Pour chaque effet de transformation cité ci-dessus, nous allons soit pouvoir utiliser les fonctions scale3d(), translate3d() et rotate3d(), soit utiliser les notations longues scaleX(), scaleY(), etc. et en les complétant avec scaleZ(), translateZ() et rotateZ().

Voici deux exemples de translation et de rotation 3D qu'on va effectuer durant une animation :