
# Les bases en CSS

| | | | |
|---|---|---|---|
| Rédigé par: Jean-Luc Colson | | | |
| Date première rédaction: Jan 2020 | | | |
| | | | |
| | | | |
| SUIVI DES MODIFICATIONS A LA FICHE | | | |
| Série | Date | Page(s) modifiée(s) | Raison |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Contents

# Sélecteurs et propriétés CSS

Le CSS est un langage qui a été inventé pour styliser les contenus de nos pages en leur appliquant des styles.

Dans cette nouvelle partie, nous allons passer en revue les notions de base du CSS en comprenant notamment les grands principes de fonctionnement de ce langage et en apprenant à cibler des contenus de manière précise pour pouvoir leur appliquer des styles.

## Les sélecteurs CSS : définition

Pour pouvoir appliquer un style à un contenu, il va déjà falloir le cibler, c'est-à-dire trouver un moyen d'indiquer qu'on souhaite appliquer tel style à un contenu en particulier.

Pour cela, nous allons utiliser des sélecteurs. Les sélecteurs sont l'un des éléments fondamentaux du CSS.

De manière très schématique et très simplifiée, nous allons utiliser nos sélecteurs en CSS pour cibler des contenus HTML et leur appliquer des styles.

Il existe différents types de sélecteurs en CSS : certains sélecteurs vont s'appuyer sur le nom des éléments, comme le sélecteur CSS p par exemple qui va servir à cibler tous les éléments p d'une page. Ce type de sélecteurs est appelé « sélecteur d'éléments » tout simplement car ils vont être identiques aux éléments HTML sélectionnés ou encore « sélecteurs simples ».

D'autres sélecteurs, en revanche, vont être plus complexes et nous permettre de sélectionner un élément HTML en particulier ou un jeu d'éléments HTML en fonction de leurs attributs ou même de leur état : on va ainsi pouvoir appliquer des styles à un élément uniquement lorsque la souris de l'utilisateur passe dessus par exemple.

N'essayez pas forcément de comprendre immédiatement le code ci-dessus : le but n'est ici que de vous fournir un exemple concret de ce qu'on va pouvoir faire en CSS. Nous allons apprendre à utiliser la majorité des sélecteurs CSS et notamment les plus courants et les plus utiles dans la suite de ce cours.

## Les propriétés CSS : définition

Les propriétés vont nous permettre de choisir quel(s) aspect(s) (ou "styles") d'un élément HTML on souhaite modifier.
Par exemple, nous allons pouvoir modifier la couleur d'un texte et lui appliquer la couleur que l'on souhaite grâce à la propriété color (« couleur », en français).

Une propriété va être accompagnée d'une ou plusieurs valeurs qui vont définir le comportement de cette propriété.
Par exemple, la propriété color peut prendre le nom d'une couleur (en anglais). Si l'on donne la valeur red (rouge) à notre propriété color, les textes au sein des éléments HTML auxquels on applique cette propriété s'afficheront en rouge.

## Les déclarations CSS : premier exemple pratique

Prenons immédiatement un premier exemple ensemble en expliquant bien à quoi correspond chaque élément du code afin d'illustrer ce que nous venons de dire et de bien voir comment le CSS fonctionne.

Je vous demande pour le moment de ne pas vous soucier des questions pratiques concernant la liaison entre les codes HTML et CSS mais simplement de vous concentrer sur le code CSS présenté.

```
1  p{
2      color: blue;
3      border: 2px solid orange;
4      padding: 5px;
5  }
```

Détaillons le code CSS ci-dessus. Ici, nous utilisons le sélecteur CSS simple p pour cibler tous les paragraphes de nos pages HTML. Ensuite, nous ouvrons une paire d'accolades. Entre ces accolades, nous allons préciser les différents styles que l'on souhaite appliquer à nos éléments p.

En l'occurrence, on définit une couleur, bordure et une marge interne personnalisées pour tous nos paragraphes grâce aux propriétés CSS color, border et padding.

Le texte de nos paragraphes va donc s'afficher en bleu et nos paragraphes auront des bordures solides oranges de 2px d'épaisseur et des marges internes de 5px.

Le couple « propriété : valeur » est appelée « déclaration » en CSS. Chaque déclaration doit se terminer par un point-virgule.

On va pouvoir écrire autant de déclarations que l'on souhaite à l'intérieur du couple d'accolades qui suit un sélecteur en CSS et ainsi pouvoir définir le comportement de plusieurs propriétés facilement.

# Où écrire le code CSS ?

Avant d'étudier les mécanismes du CSS en soi, il convient de comprendre où placer le code CSS afin qu'il s'applique bien à un fichier HTML.

En effet, lorsqu'on code, pensez bien que rien n'est jamais « magique » et qu'au contraire tout le code qu'on va pouvoir écrire repose sur des règles et des mécanismes. Comprendre ces règles et ces mécanismes et notamment comment différents langages de programmation vont pouvoir fonctionner ensemble est certainement l'une des choses les plus complexes lorsqu'on est débutant.

Pour cela, je pense qu'il ne faut pas essayer de tout comprendre tout de suite : c'est tout à fait normal s'il y a des mécanismes dont vous ne comprenez pas tous les rouages immédiatement. Avec un peu de temps, de la pratique et de nouvelles connaissances sur la programmation les choses pas claires au début devraient devenir de plus en plus évidentes.

Dans le cas présent, nous avons notre code HTML d'un côté et nous aimerions lui appliquer des styles en CSS. Cependant, il va falloir d'une manière ou d'une autre « lier » notre code CSS à notre code HTML afin que les éléments de nos pages HTML tiennent bien compte des styles qu'on a voulu leur appliquer en CSS.

Pour faire cela, nous allons pouvoir écrire le code CSS à trois endroits différents. Chaque méthode va présenter des avantages et des inconvénients selon une situation donnée et c'est le sujet que nous allons aborder dans cette leçon.

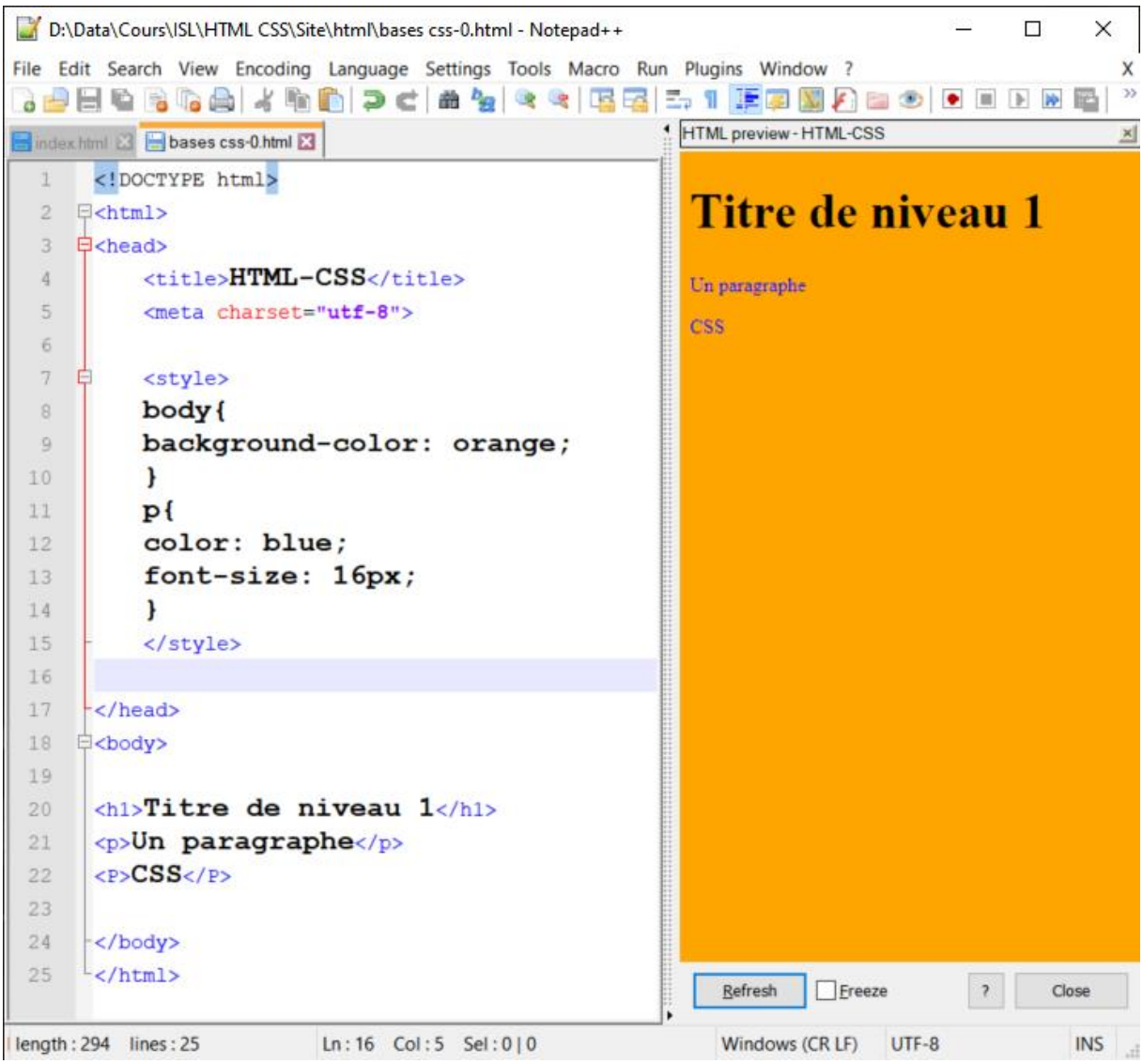
## Méthode n°1 : écrire le CSS au sein du fichier HTML, dans un élément style

La première façon d'écrire du code CSS va être à l'intérieur même de notre page HTML, au sein d'un élément style.

En plaçant le CSS de cette façon, le code CSS ne s'appliquera qu'aux éléments de la page HTML dans laquelle il a été écrit.

Cette première méthode d'écriture du CSS n'est pas recommandée, pour des raisons de maintenance et d'organisation du code en général. Cependant, elle peut s'avérer utile pour modifier rapidement les styles d'une page HTML ou si vous n'avez pas facilement accès aux fichiers de style de votre site. Nous voyons donc cette première méthode à titre d'exemple, afin que vous sachiez l'identifier si un jour vous voyez du code CSS écrit de cette façon dans un fichier et que vous puissiez l'utiliser si vous n'avez pas d'autre choix.

Nous allons devoir ici placer notre élément style au sein de l'élément head de notre fichier HTML. Voici comment on va écrire cela en pratique :



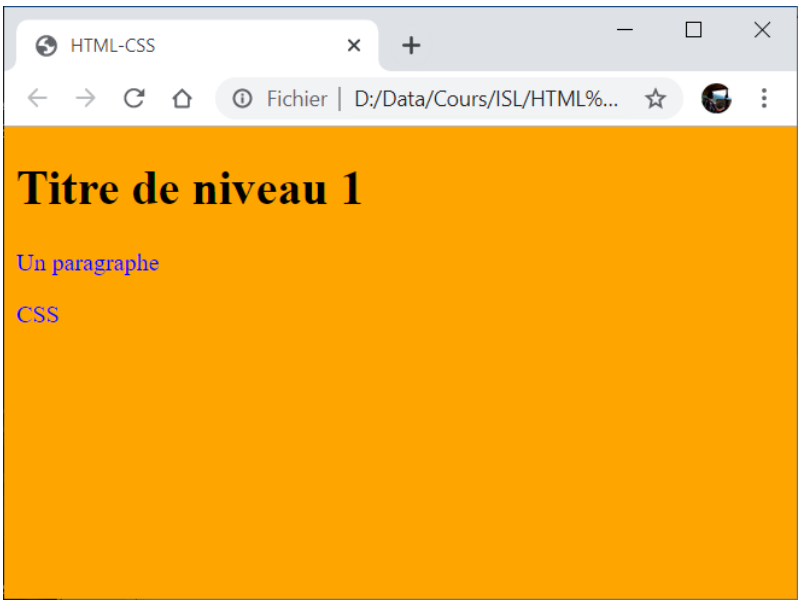

Ici, nous créons un fichier HTML tout à fait classique contenant un titre h1 et deux paragraphes.

Nous voulons ensuite rajouter des styles à notre page. Pour cela, nous plaçons un élément style dans l'élément head de notre page. Nous allons déclarer nos styles CSS au sein de cet élément.

Dans mon code CSS, je commence par cibler l'élément body avec le sélecteur élément du même nom et je définis une couleur de fond (background-color) orange pour cet élément. Comme l'élément body représente toute la partie visible de ma page, le fond de la page entière sera orange.

Ensuite, je définis également une couleur bleue pour le texte de mes paragraphes ainsi qu'une taille de police d'écriture de 16px.

Voici le résultat obtenu :



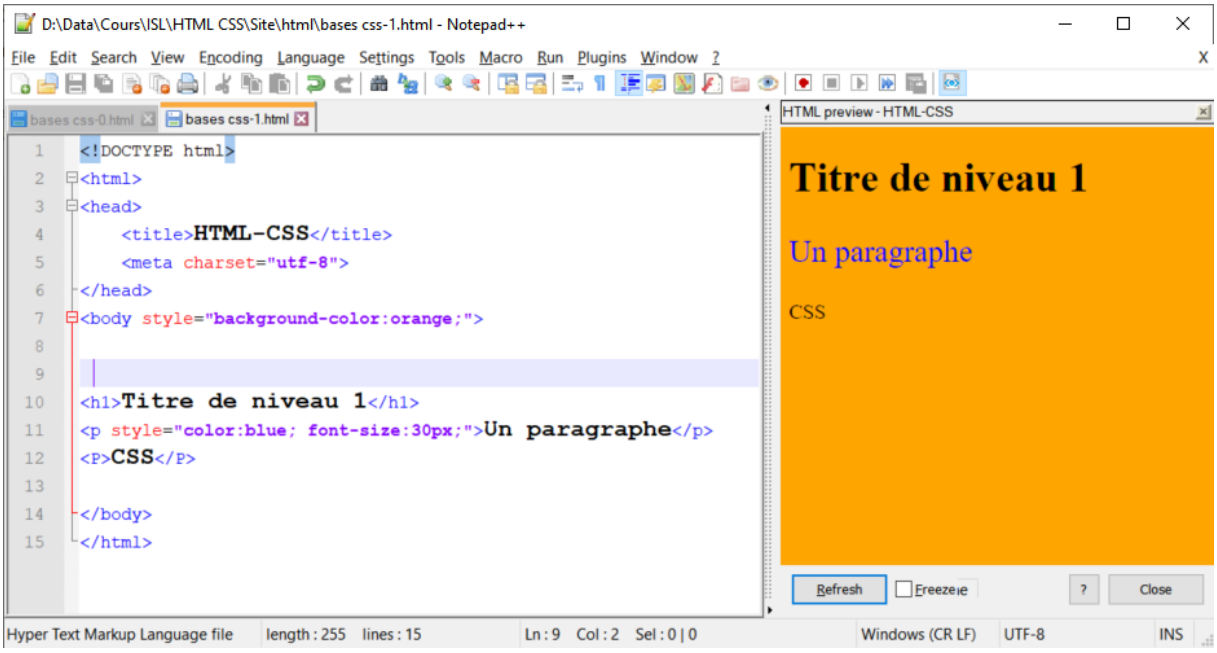

## Méthode n°2 : déclarer le CSS au sein du fichier HTML, dans des attributs style

Nous pouvons également écrire notre code CSS au sein d'attributs style qu'on va ajouter à l'intérieur de la balise ouvrante des éléments HTML pour lesquels on souhaite modifier les styles.

Nous allons passer en valeurs des attributs style des déclarations CSS pour modifier certains styles précis de l'élément HTML en question. En effet, en utilisant cette méthode, les styles déclarés dans un attribut style ne vont s'appliquer qu'à l'élément dans lequel ils sont écrits, et c'est la raison pour laquelle nous n'allons pas avoir besoin de préciser de sélecteur ici.

Attention à ne pas confondre les attributs style qu'on va devoir placer au sein de la balise ouvrante de chaque élément dont on souhaite modifier les styles avec l'élément style qu'on va placer dans l'élément head de nos fichiers HTML.

Dans l'exemple ci-dessous, on applique à nouveau une couleur de fond orange à notre élément body ainsi qu'une couleur bleue et une taille de 20px au texte de notre premier paragraphe uniquement :




Cette deuxième méthode d'écriture du CSS, bien qu'elle puisse sembler pratique à priori puisqu'elle permet de n'appliquer des styles qu'à un élément en particulier plutôt qu'à tous les éléments d'un même type n'est également pas recommandée et est à éviter tant que possible pour des raisons de maintenabilité et de performance du code.

En effet, déclarer nos styles comme cela n'est vraiment pas efficient puisque cela va demander énormément d'écriture et également énormément de temps de réécriture le jour où l'on souhaite modifier des styles.

Pas d'inquiétude : nous allons apprendre à cibler précisément un élément ou un groupe d'éléments en particulier pour leur appliquer des styles personnalisés plus tard dans ce cours.

## Méthode n°3 : écrire le CSS dans un fichier séparé

Finalement, nous pouvons écrire notre code CSS dans un fichier séparé portant l'extension « .css ». C'est la méthode recommandée, qui sera utilisée autant que possible.

Cette méthode comporte de nombreux avantages, notamment une meilleure maintenabilité du code grâce à la séparation des différents langages, ainsi qu'une meilleure lisibilité.

Cependant, le plus gros avantage de cette méthode est qu'on va pouvoir appliquer des styles à plusieurs pages HTML en même temps, d'un seul coup.

En effet, en utilisant l'une des deux premières méthodes, nous aurions été obligés de réécrire tout notre code CSS pour chaque page HTML (ou même pour chaque élément !) composant notre site puisque les codes CSS étaient déclarés dans une page ou dans un élément spécifique et ne pouvaient donc s'appliquer qu'à la page ou qu'à l'élément dans lesquels ils étaient déclarés.

De plus, en cas de modification, il aurait également fallu modifier chacune de nos pages à la main, ce qui n'est pas viable pour un site de taille moyenne qui va être composé de quelques centaines de pages.

En déclarant notre code CSS dans un fichier séparé, au contraire, nous allons pouvoir utiliser le code de ce fichier CSS dans autant de fichiers HTML qu'on le souhaite, en indiquant aux différents fichiers HTML qu'ils doivent appliquer les styles contenus dans ce fichier CSS. Ainsi, lorsque nous voudrons modifier par exemple la couleur de tous les paragraphes de nos pages HTML nous n'aurons qu'à modifier la déclaration relative dans le fichier CSS.

Voyons immédiatement comment mettre cela en place en pratique. Pour cela, nous allons commencer par créer un nouveau fichier dans notre éditeur qu'on va appeler cours.css. Nous allons enregistrer ce fichier et le placer dans le même dossier que notre page HTML pour plus de simplicité.

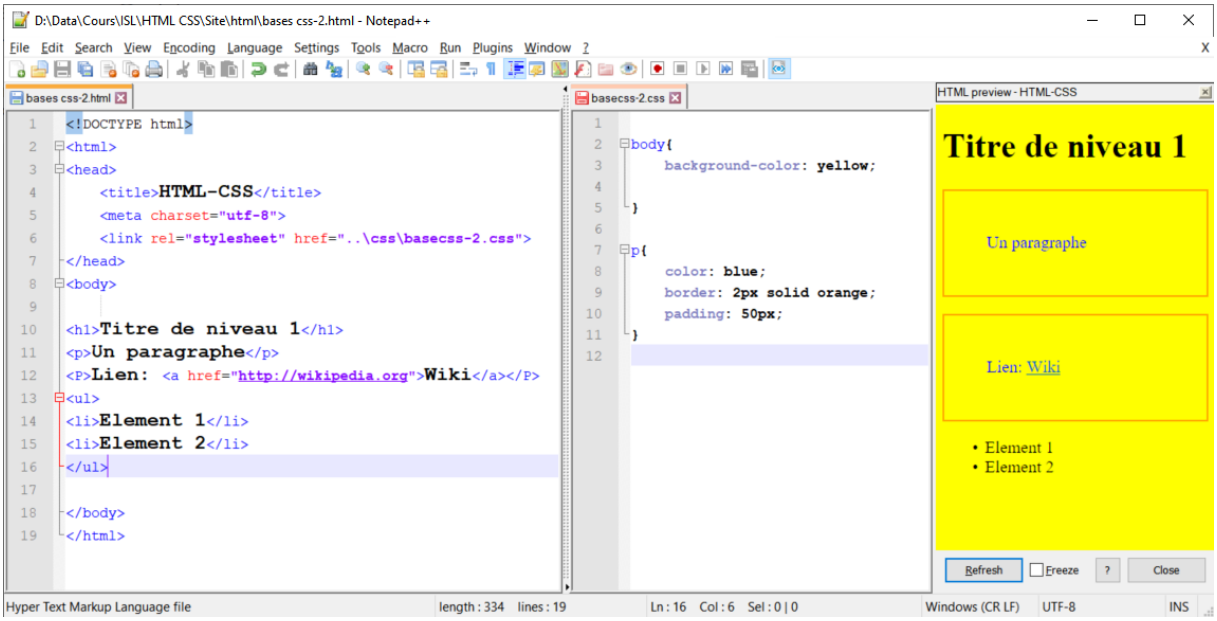
Nous travaillons donc dorénavant avec deux fichiers : un fichier appelé cours.html et un fichier cours.css.

Il va donc maintenant falloir « lier » notre fichier HTML à notre fichier CSS pour indiquer au navigateur qu'il doit appliquer les styles contenus dans le fichier cours.css à notre fichier cours.html.

Pour cela, nous allons utiliser un nouvel élément HTML : l'élément link (« lien », en français). On va placer l'élément link au sein de l'élément head de notre fichier HTML. Cet élément se présente sous la forme d'une balise orpheline et va avoir besoin de deux attributs pour fonctionner correctement :

- Un attribut rel qui va nous servir à préciser le type de ressource que l'on souhaite lier à notre fichier HTML. Dans notre cas, nous indiquerons la valeur stylesheet pour « feuille de style » ;
- Un attribut href qui va indiquer l'adresse relative de la ressource que l'on souhaite lier par rapport à l'emplacement de notre fichier HTML. Ici, comme nous avons enregistré nos deux fichiers dans le même dossier, il suffira d'indiquer le nom de notre fichier CSS en valeur de href.

Nos deux fichiers sont maintenant liés et les styles déclarés dans notre fichier CSS vont bien être appliqués aux éléments de notre page HTML.


D:\Data\Cours\ISL\HTML CSS\Site\html\bases css-2.html - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
bases css-2.html
basecss-2.css
HTML preview - HTML-CSS
Titre de niveau 1
Un paragraphe
Lien: Wiki
Element 1
Element 2
Refresh
Freeze
Close
Hyper Text Markup Language file
length : 334 lines : 19
Ln : 16 Col : 6 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS

# Commentaires et indentation en CSS

Les commentaires ne sont pas des éléments spécifiques au langage HTML. En réalité, la grande majorité des langages de programmation permettent aux développeurs de commenter leur code via des syntaxes différentes propres à chaque langage car commenter est reconnu comme une bonne pratique en programmation et se révèle souvent indispensable ou à minima très utile.

Dans cette nouvelle leçon, nous allons donc voir comment commenter en CSS et également discuter de l'indentation en CSS.

## Commenter le code CSS

Tout comme nous avons vu qu'on pouvait écrire des commentaires en HTML, nous allons également pouvoir commenter notre code CSS.

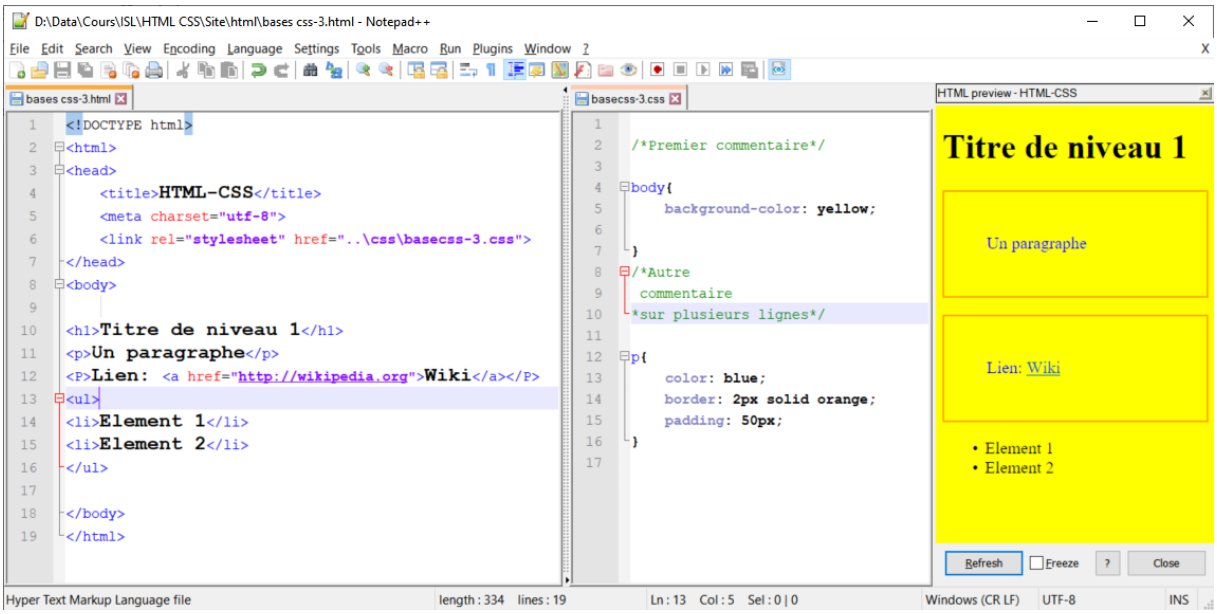
Les commentaires n'influent une nouvelle fois en rien sur le code et ne sont pas visibles par les utilisateurs.

Commenter le code CSS n'est pas une option : cela va très vite devenir indispensable car vous allez vous rendre compte que les fichiers CSS s'allongent très vite.

Il est donc essentiel de bien organiser et de bien commenter son code CSS afin de ne pas faire d'erreur en appliquant par exemple deux styles différents à un même élément.

Le CSS, tout comme le HTML et à la différence d'autres langages de développement ne possède qu'une seule syntaxe qui va nous permettre de créer à la fois des commentaires mono-ligne et multi-lignes.

Cette syntaxe est la suivante : /*Un commentaire CSS*/. Regardez plutôt l'exemple ci-dessous :



Dans l'exemple ci-dessus, notez que les étoiles en début de ligne pour mon commentaire multi-lignes ne sont absolument pas nécessaires (à part pour la première ligne, évidemment) : ce n'est que de la décoration afin de bien voir que l'on commente.

Vous pouvez également remarquer une utilisation intéressante des commentaires et qui est très commune en CSS : le fait de commenter une déclaration CSS.

En effet, vous voudrez parfois supprimer momentanément une déclaration CSS, pour effectuer des tests par exemple. Plutôt que de l'effacer complètement, vous pouvez la commenter.

Ainsi, la déclaration CSS ne sera plus prise en compte. Vous n'aurez ensuite plus qu'à enlever le commentaire pour la « réactiver ».

## Indenter en CSS

Indenter en CSS est également très important afin de conserver le plus de clarté possible dans son code et de paraître professionnel si un jour vous devez le distribuer.

En termes de règles, nous indenterons en général d'une tabulation les différentes déclarations concernant un sélecteur donné.

Pour plus de lisibilité, nous retournerons également à la ligne après chaque déclaration. Notez que cela augmentera de façon très minime le temps d'exécution du code et donc le temps d'affichage de la page.

Cependant, en phase de développement tout au moins, il est essentiel de conserver un code aéré et propre. Vous pourrez toujours le compresser par la suite ; de nombreux outils existent sur le web pour cela.

# Sélecteurs CSS simples et combinateurs

Le CSS va nous permettre de mettre en forme nos contenus HTML en appliquant des styles aux différents éléments. Cependant, pour appliquer un style particulier à un ou plusieurs éléments HTML en CSS, il va avant tout falloir les cibler, c'est-à-dire indiquer avec précision à quels éléments doivent s'appliquer les styles créés en CSS.

Le but de cette leçon est d'apprendre à se servir de quelques sélecteurs CSS « simples » et de comprendre leur limitation. Nous allons également en profiter pour définir plus précisément les différents types de sélecteurs CSS et directement voir comment combiner différents sélecteurs simples pour en créer des plus complexes.

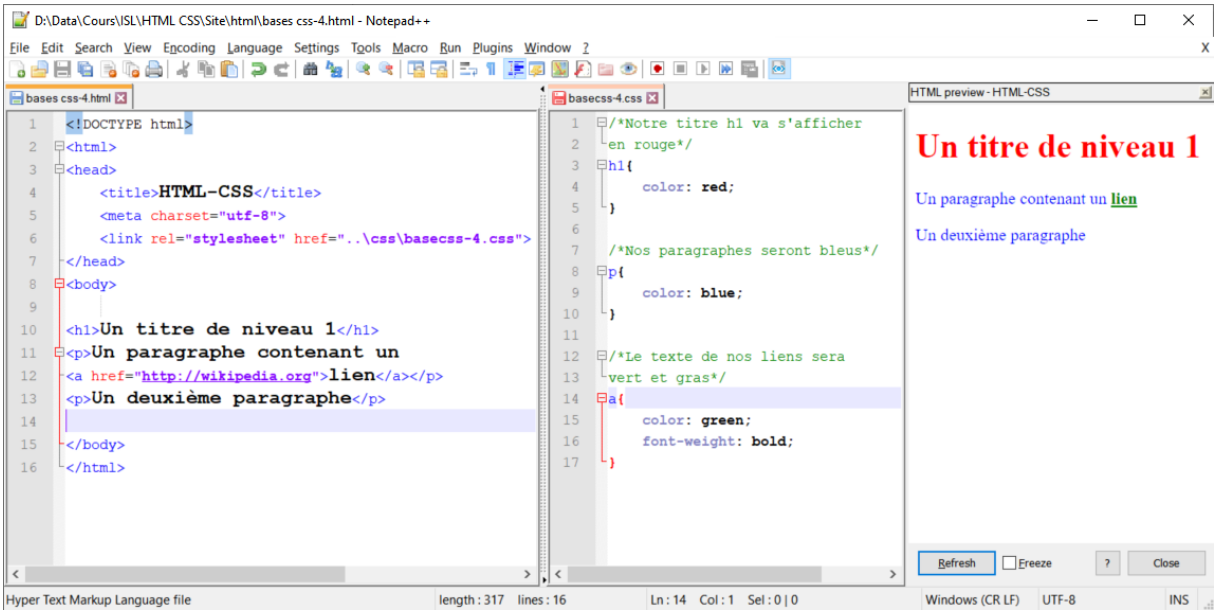
## Les sélecteurs CSS éléments ou sélecteurs « simples »

Il existe de nombreux types de sélecteurs CSS et autant de moyens de cibler des contenus HTML en CSS.

La manière la plus simple de cibler un type d'éléments HTML en CSS est néanmoins d'utiliser des sélecteurs éléments ou sélecteurs « simples ». Ces sélecteurs sont appelés « sélecteurs éléments » tout simplement car ils reprennent le nom des éléments HTML qu'ils sélectionnent.

Par exemple, le sélecteur CSS p va cibler tous les éléments p (c'est-à-dire tous les paragraphes) d'une page HTML.

De même, le sélecteur CSS h1 va nous permettre d'appliquer des styles à notre titre h1 ; le sélecteur a va nous permettre de mettre en forme nos liens, etc.



## Comprendre les limitations des sélecteurs CSS simples

L'utilisation de sélecteurs simples doit être favorisée tant que possible car ces sélecteurs consomment moins de ressources que des sélecteurs plus complexes et car ils sont plus clairs.

Ceci étant dit, nous n'allons bien souvent pas pouvoir nous contenter de n'utiliser que des sélecteurs simples car ceux-ci vont considérablement limiter nos options de ciblage et car ils ne vont pas nous permettre d'exploiter toute la puissance du CSS.

En effet, en utilisant uniquement les sélecteurs éléments, nous allons être obligés d'appliquer les mêmes styles à tous les éléments d'un même type ce qui n'est pas très flexible.

Comment appliquer des styles à un élément en particulier ou à plusieurs éléments différents choisis ? Pour faire cela, nous allons devoir utiliser des sélecteurs complexes.

## Introduction aux sélecteurs CSS complexes et combinateurs

Toute la puissance du CSS réside dans les options que nous offre ce langage pour cibler précisément un contenu HTML dans une page.

En effet, en plus des sélecteurs simples, le CSS met à notre disposition une panoplie de sélecteurs que l'on va pouvoir utiliser pour cibler des contenus de manière très précise :

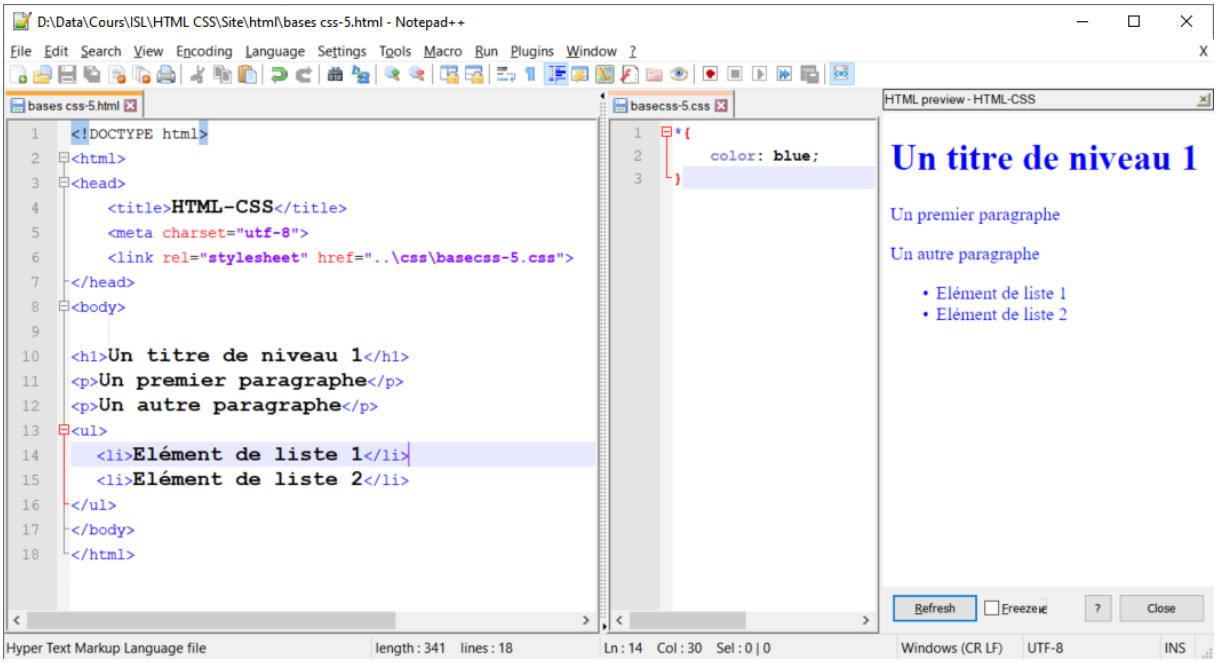
- On va pouvoir utiliser des sélecteurs CSS combinateurs qui vont être en fait la combinaison de plusieurs sélecteurs simples à l'aide de caractères spéciaux à la signification précise ;
- On va pouvoir cibler des contenus HTML selon le fait qu'ils possèdent un certain attribut ou même selon la valeur d'un attribut ;
- On va pouvoir utiliser les pseudo classes qui vont nous permettre d'appliquer des styles à des éléments en fonction de leur état, c'est-à-dire en fonction des actions d'un utilisateur (contenu cliqué, coché, visité, etc.), de la place de l'élément dans le document, etc. ;
- On va pouvoir utiliser les pseudo éléments qui vont nous permettre de n'appliquer des styles qu'à certaines parties des éléments.

Nous allons apprendre à faire tout cela au cours de ce cours. Pour le moment, toutefois, nous allons nous contenter de présenter et d'apprendre à manipuler les différents caractères « combinateurs » qui vont nous permettre de combiner des sélecteurs CSS simples afin d'en créer des plus complexes.

## Sélectionner tous les éléments avec le sélecteur CSS universel ou sélecteur étoile (*)

Le sélecteur CSS étoile * ne nous permet pas à proprement parler de combiner différents sélecteurs simples entre eux mais permet de sélectionner tous les éléments HTML d'une page d'un coup ; c'est pourquoi il est également appelé sélecteur CSS universel.

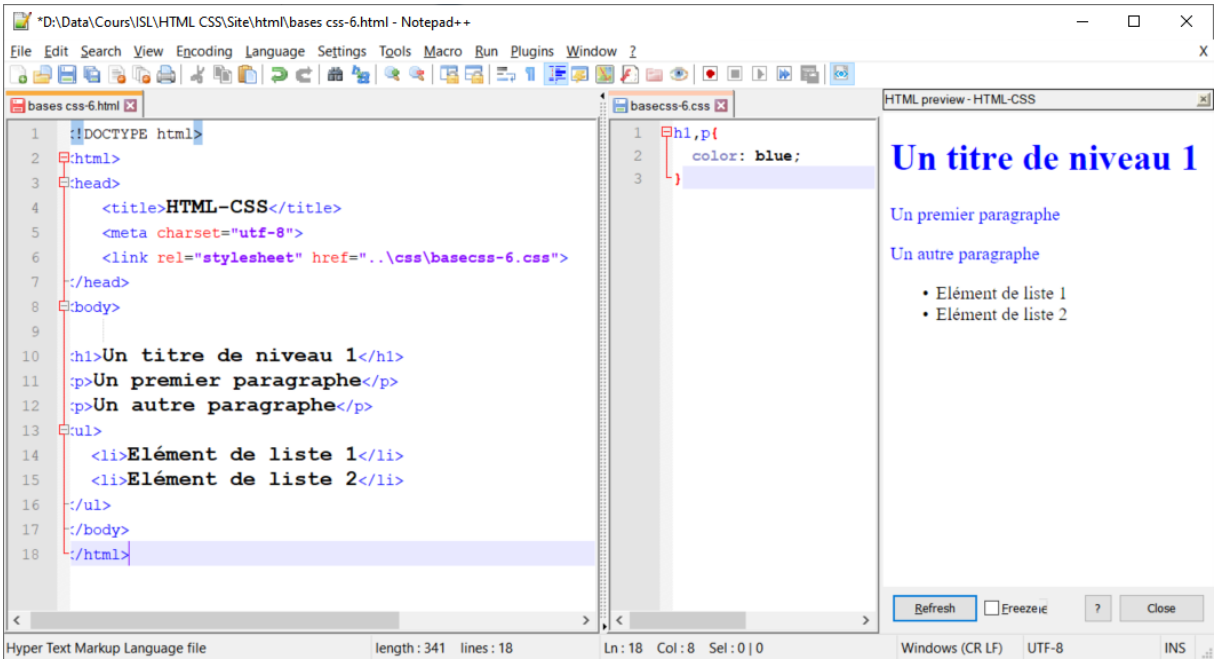
Ce sélecteur va donc nous permettre d'appliquer les mêmes styles à tous les éléments d'une page. Cela peut être très utile pour par exemple définir une police par défaut ou effectuer un reset des marges de tous les éléments pour ensuite les positionner plus précisément.




## Appliquer des styles à plusieurs éléments avec le caractère virgule (,)

Pour appliquer un même style à deux types éléments différents sans avoir à recopier le style deux fois en CSS, nous allons simplement pouvoir séparer nos deux sélecteurs par une virgule. Les styles CSS déclarés juste après s'appliqueront ainsi aux deux éléments ou groupes d'éléments sélectionnés.

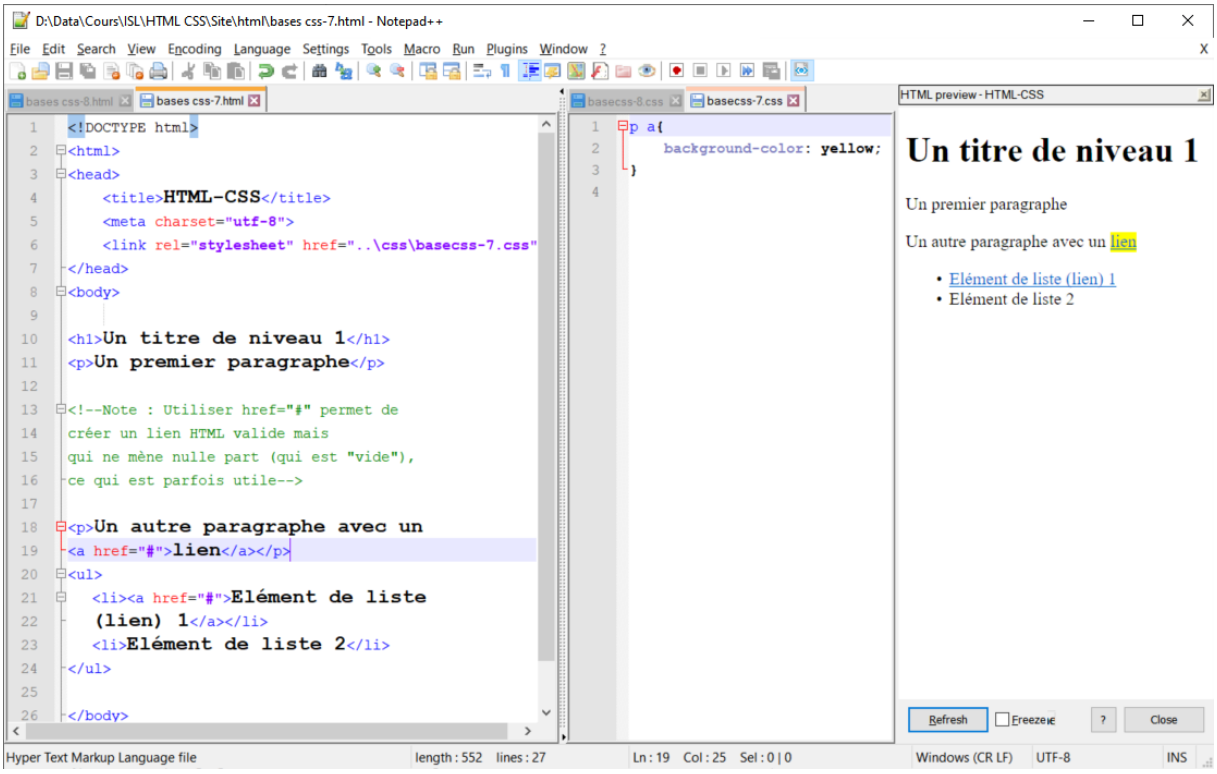
Bien évidemment, rien ne nous empêche d'appliquer un même style à 3, 4, … éléments ou groupes d'éléments. Le tout est de séparer les différents sélecteurs par des virgules.




## Utiliser plusieurs sélecteurs CSS à la suite

En mentionnant plusieurs sélecteurs à la suite en CSS, nous allons pouvoir appliquer des styles à certains éléments contenus dans d'autres éléments.

Par exemple, utiliser le sélecteur p a en CSS va nous permettre d'appliquer des styles à tous les éléments a contenus dans des éléments p et seulement aux éléments a contenus dans des éléments p

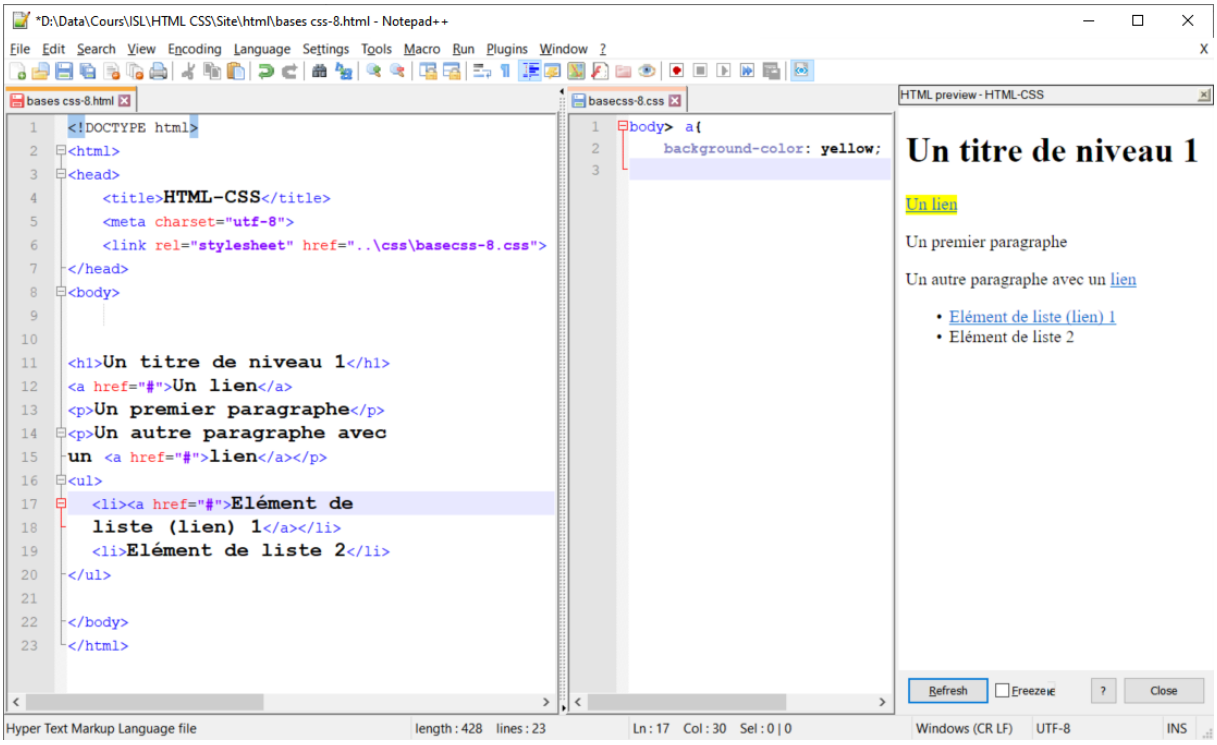



## Appliquer des styles aux enfants directs d'un autre élément

Nous allons également pouvoir cibler uniquement un élément ou un groupe d'éléments enfants directs d'un autre élément en utilisant le signe de supériorité stricte ou le caractère chevron fermant >.

Un élément est un enfant direct ou « descendant direct » d'un autre élément s'il est directement contenu dans celui-ci.

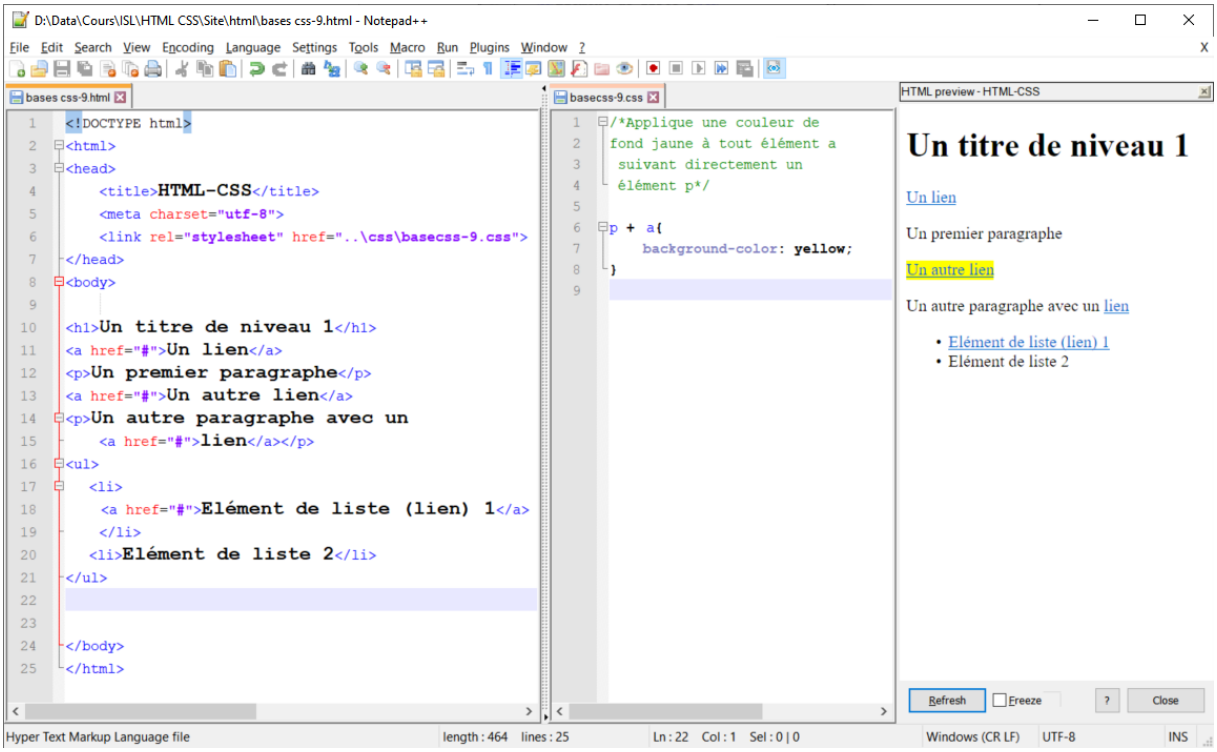
Par exemple, nous allons pouvoir appliquer des styles à tous les liens (éléments a) qui sont des enfants directs de l'élément body et uniquement à ceux-ci :




## Sélectionner l'élément suivant directement un autre élément en CSS

Le CSS va nous permettre de cibler encore plus précisément un élément en ciblant l'élément suivant directement un autre élément grâce au caractère +.

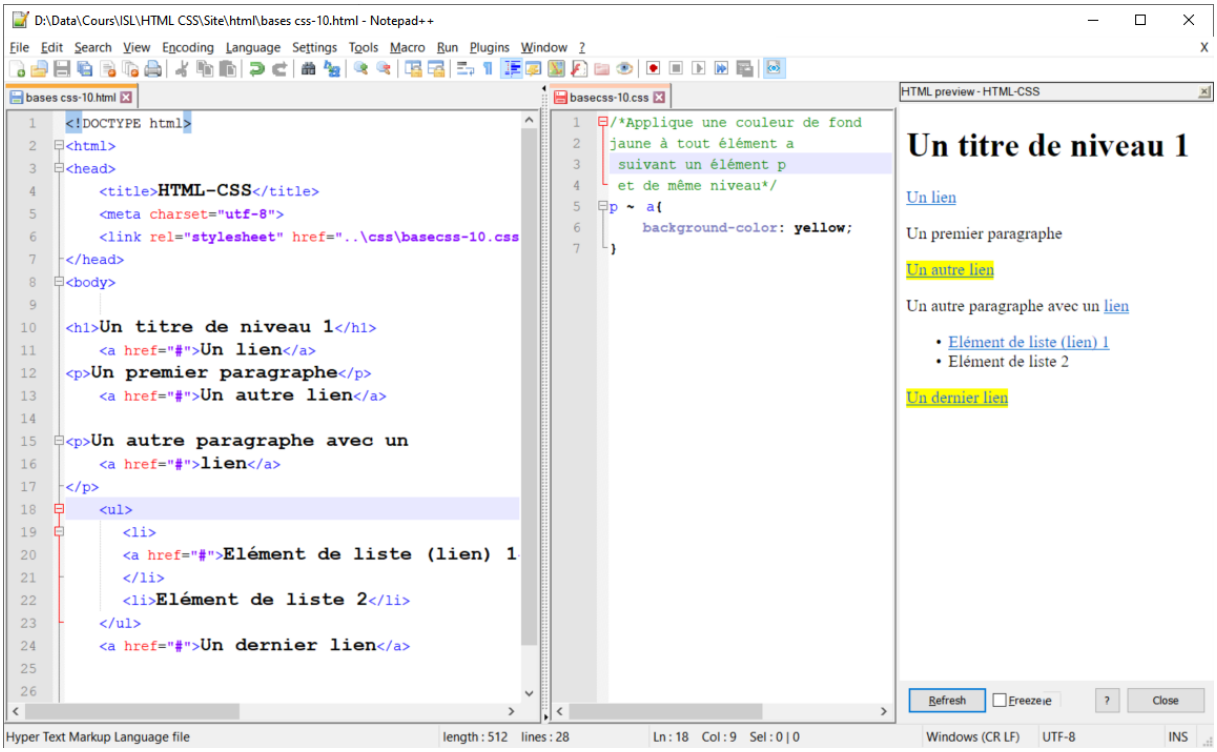
Par exemple, on va pouvoir cibler les éléments a (liens) suivant directement un élément p :




## Sélectionner tous les éléments suivant un autre élément en CSS

Le caractère ~ va nous être plus permissif que le caractère + en nous permettant cette fois-ci de sélectionner tous les éléments déclarés après un autre élément en HTML de même niveau (c'est-à-dire possédant le même parent direct).

Par exemple, on va pouvoir cibler tous les éléments a placés après un élément p et qui sont de même niveau :




Ici, nos styles CSS ne s'appliquent pas à aux deux liens dans notre paragraphe et dans notre élément de liste car ceux-ci ne sont pas au même niveau que les différents paragraphes de notre page : ils sont inclus dans d'autres éléments et ne possèdent donc pas le même parent qu'un autre paragraphe pouvant les précéder.

Résumé des caractères spéciaux permettant de combiner des sélecteurs et signification

Vous pourrez trouver ci-dessous un résumé des différents caractères vus dans cette partie et qui vont nous permettre de combiner des sélecteurs CSS simples afin d'en créer des plus complexes :

| **Sélecteur CSS** | **Signification** |
|---|---|
| * | Sélectionne tous les éléments |
| E, F | Sélectionne tous les éléments de type E et de type F |
| E F | Sélectionne tous les éléments F à l'intérieur des éléments E |
| E > F | Sélectionne les éléments F enfants directs des éléments E |
| E + F | Sélectionne tout élément F placé directement après un élément E |

| Sélecteur CSS | Signification |
|---|---|
| E~F | Sélectionne tout élément F placé après un élément E dans la page |

# Les attributs HTML class et id et les sélecteurs CSS associés

Cette leçon est consacrée à la découverte et à l'utilisation des attributs HTML class et id.

Nous allons pouvoir ajouter ces deux attributs à n'importe quel élément HTML.

Ces deux attributs sont particuliers en HTML puisqu'ils n'ont pas été créés pour préciser le fonctionnement d'un élément HTML en particulier (ce qui est normalement le rôle de tout attribut HTML) mais vont être principalement utilisés pour cibler certains éléments HTML et leur appliquer des styles en CSS.

## Présentation des attributs HTML class et id et cas d'utilisation

Les attributs HTML class et id sont des attributs dits globaux car on va pouvoir les ajouter dans la balise ouvrante de n'importe quel élément HTML.

Ces deux attributs vont être principalement utilisés dans un but de mise en forme : ils vont nous servir à appliquer des styles CSS aux éléments qui vont les contenir.

En effet, à la différence d'un attribut href par exemple, les attributs class et id ne vont pas servir à préciser le fonctionnement d'un élément HTML mais vont simplement être très utiles pour cibler un élément précisément.

Nous allons effectivement très facilement pouvoir nous resservir de ces deux attributs en CSS grâce aux sélecteurs associés .class et #id.

Donc :

| **html** | **css** |
|---|---|
| class="classetoto" | .classetoto{…} |
| id="idtoto" | #idtoto{…} |

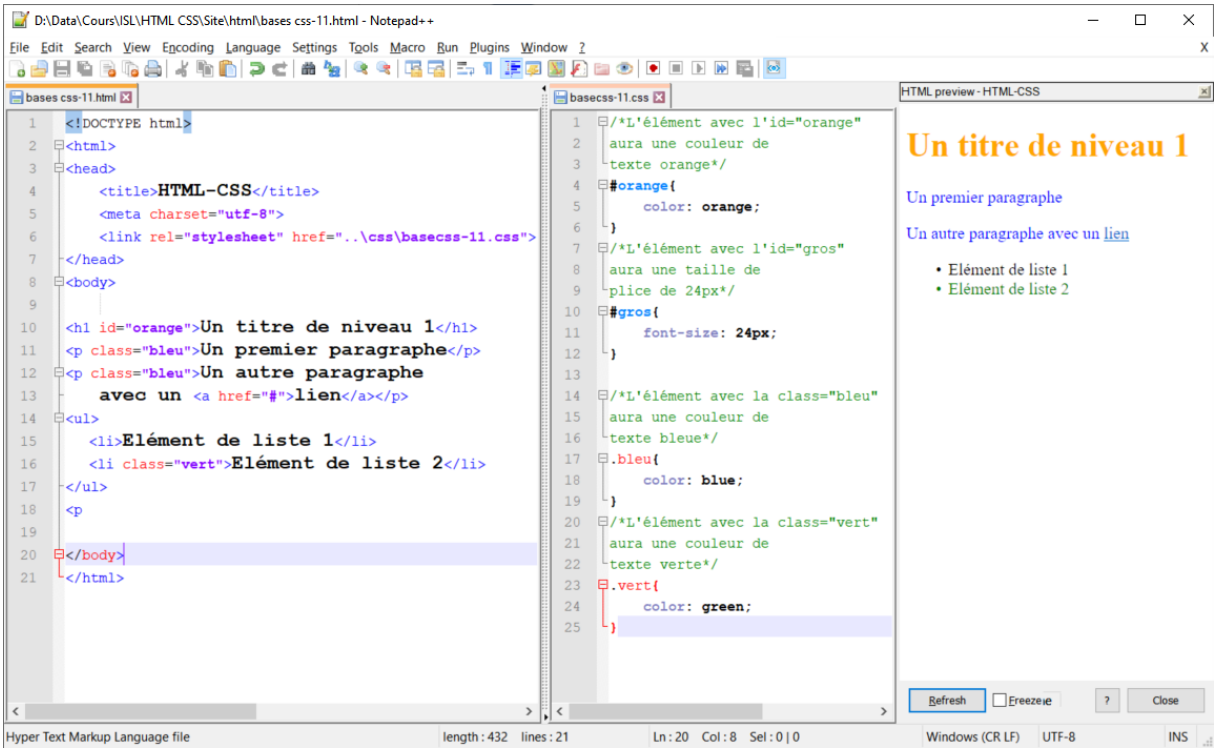
## Premier exemple d'utilisation des attributs HTML class et id et des sélecteurs CSS associés

Voyons immédiatement de manière pratique comment vont fonctionner les attributs class et id et comment on va pouvoir les utiliser en CSS pour cibler et appliquer des styles particuliers à des éléments choisis.

Pour cela, nous allons créer une page HTML et allons placer des attributs class et id dans différents éléments.

Nous allons déjà devoir renseigner une valeur pour chaque attribut class et id. Les valeurs indiquées pour les attributs ne doivent contenir ni caractères spéciaux ni espaces et commencer par une lettre. En pratique, on essaiera d'attribuer des valeurs qui font sens à nos différents attributs.

Notez déjà que chaque id doit avoir une valeur propre ou unique dans une page. En revanche, on va pouvoir attribuer la même valeur à plusieurs attributs class différents.




Ici, on ajoute un attribut id="orange" dans la balise ouvrante de notre élément h1 et un attribut id="gros" à notre dernier paragraphe. On ajoute également un même attribut class="bleu" à nos deux premiers paragraphes et un attribut class="vert" à un élément de notre liste.

Ensuite, on va lier des styles CSS à ces différents id et class en utilisant les sélecteurs CSS associés.

Pour cibler un id en particulier en CSS, on utilisera le symbole dièse # suivi de la valeur de l'id auquel on souhaite lier des styles.

Pour cibler une class en particulier en CSS, on utilisera le symbole point . suivi de la valeur de la class à laquelle on souhaite lier des styles.

## Class vs id : Quelles différences et quel attribut utiliser ?

Il existe une différence notable entre les deux attributs class et id : chaque id doit avoir une valeur unique dans une même page tandis que plusieurs attributs class peuvent partager la même valeur.

Cela fait que l'attribut id est beaucoup plus spécifique que l'attribut class et que ces deux attributs vont avoir des rôles et buts différents notamment pour la mise en forme CSS.

## Utilisation des classes en HTML et en CSS

Ainsi, nous utiliserons généralement des attributs class pour définir des styles généraux et communs à plusieurs éléments dans une même page. Comme nous pouvons donner une même class à

plusieurs éléments, ils hériteront tous des mêmes styles sauf en cas de conflit (c'est-à-dire dans le cas où le comportement d'une même propriété a déjà été défini en CSS) bien évidemment.

Toute la puissance des attributs class et du sélecteur CSS associé va résider dans le fait qu'on va tout à fait pouvoir définir des styles CSS généraux liés à des sélecteurs .class avant même de commencer à écrire notre code HTML. Nous n'aurons ensuite plus qu'à fournir les attributs class à nos éléments HTML lors de la création de la page.

L'idée va être la suivante : créer des styles CSS et les attacher à des sélecteurs .class à priori puis attribuer les attributs class relatifs à certains éléments HTML choisis afin que les styles CSS correspondants leur soient appliqués.

Cette façon de procéder peut sembler étrange et "à l'envers" pour les personnes non expertes. Cependant, je vous garantis que c'est une très bonne façon de faire qui peut faire gagner énormément de temps pour un gros projet. C'est par ailleurs toute l'idée derrière l'utilisation de la librairie Bootstrap par exemple.

### Utilisation des id en HTML et en CSS

En revanche, comme chaque id doit être unique dans une page, nous utiliserons ce sélecteur pour appliquer des styles très précis et pour être sûr de ne cibler qu'un élément HTML en CSS.

C'est la raison pour laquelle on se sert des attributs id pour créer des liens de type ancre par exemple. En effet, nous sommes sûrs de lever toute ambiguïté sur la sélection avec un id car encore une fois celui-ci doit être unique.

Les sélecteurs CSS .class et #id ne possèdent donc pas le même degré de précision et ainsi n'ont pas le même ordre de priorité dans les styles attribués aux éléments en cas de conflit. Je vous rappelle ici qu'en cas de conflit sur un style en CSS ce sont les styles du sélecteur le plus précis qui seront appliqués.

L'ordre de priorité d'application des styles en CSS est le suivant (du plus prioritaire ou moins prioritaire) :

1. Les styles liés à un sélecteur #id ;
2. Les styles liés à un sélecteur .class.
3. Les styles liés à un sélecteur élément ;

## Plus d'exemples d'utilisation des attributs class et id en HTML et des sélecteurs CSS associés

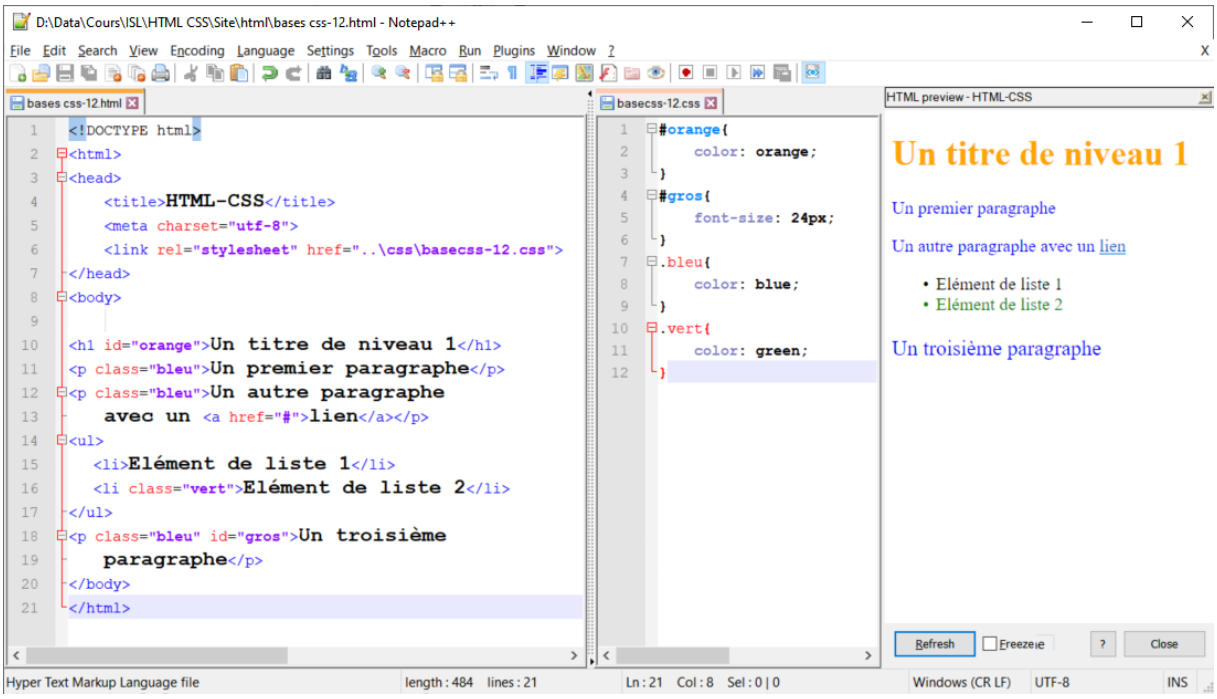
Avant tout, retenez que les valeurs indiquées pour les attributs class et id ne doivent contenir ni caractères spéciaux ni espaces et commencer par une lettre. Idéalement, nous essaierons d'utiliser des noms qui font du sens pour nos attributs class et id.

On pourra par exemple utiliser des noms relatifs aux propriétés CSS définis avec les sélecteurs associés. Faites bien attention cependant à ne pas utiliser des noms protégés c'est-à-dire des noms déjà utilisés par le HTML et qui ont déjà une signification spéciale.

## Attribuer un attribut class et un attribut id à un élément HTML

On peut tout à fait fournir plusieurs attributs à un élément HTML et notamment un attribut class et un attribut id à un élément.

Reprenons l'exemple précédent en ajoutant un attribut class à notre dernier paragraphe pour illustrer cela :



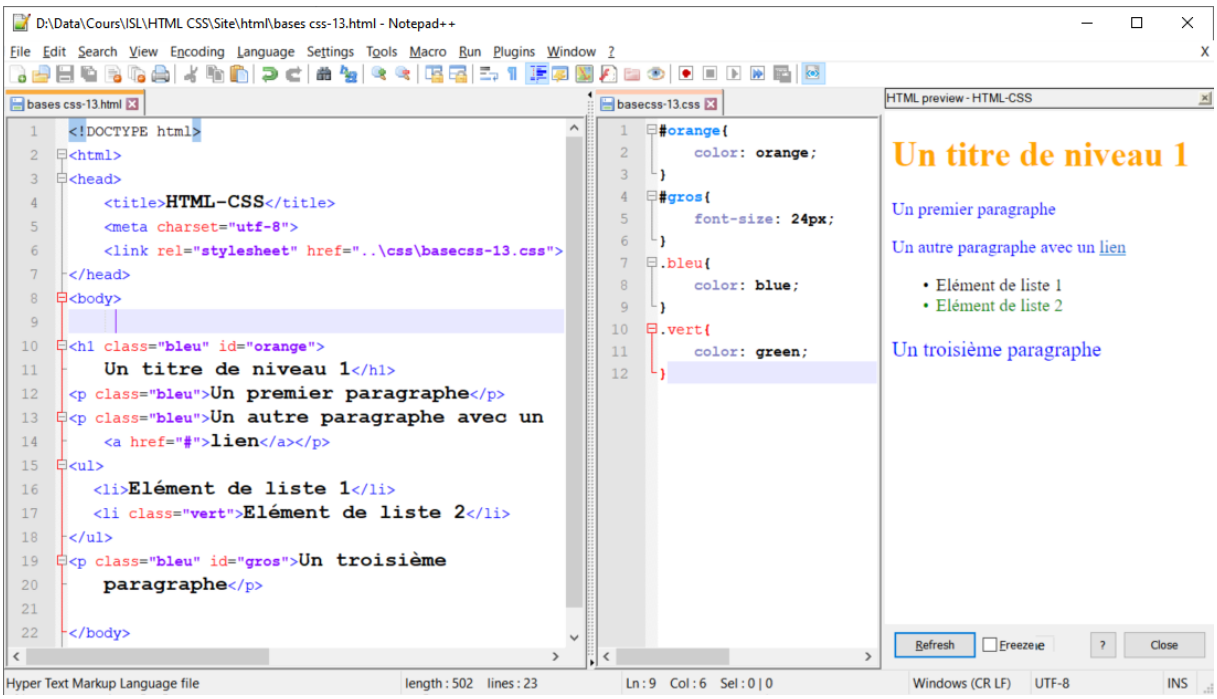

Ici, le dernier paragraphe de notre page possède à la fois un attribut class="bleu" et un id="gros". Les styles CSS liés à ces deux attributs donc être appliqués à l'élément.

Ici, nos deux attributs class="bleu" et id="gros" nous servent à appliquer des propriétés CSS différentes (color pour notre attribut class et font-size pour notre id). Il n'y a donc pas de risque de conflit.

En revanche, il y aurait eu conflit si on avait précisé des comportements différents pour la même propriété avec nos deux sélecteurs.

## Un point sur l'ordre de priorité d'application de styles CSS

Imaginons maintenant qu'on passe un attribut class et un attribut id à un même élément et qu'on définisse une même propriété CSS de manière différente pour ces id et class.




Ici on passe un attribut class="bleu" et id="orange" à notre titre h1. Or, on définit le comportement de la même propriété (la propriété color) de manière différente dans les sélecteurs .bleu et #orange.

Il y a donc conflit sur les styles.

Comme vous pouvez le voir, notre titre s'affiche en orange ce qui signifie que ce sont les styles liés à l'id qui vont être pris en compte plutôt que ceux liés à la class.

**Vous pouvez ici retenir la règle suivante dans l'application des styles CSS : ce seront toujours les styles liés au sélecteur le plus précis qui seront appliqués en cas de conflit.**

Par « précis », on entend le sélecteur qui permet d'identifier le plus précisément l'élément auxquels vont être appliqués les styles.

Ici, comme chaque id doit posséder une valeur unique dans une page, le sélecteur CSS lié à notre id est très précis et beaucoup plus précis que le sélecteur lié à l'attribut class puisqu'il permet d'identifier un élément de manière unique alors qu'un attribut class peut être partagé par plusieurs éléments et ne permet donc pas d'identifier un élément en particulier.

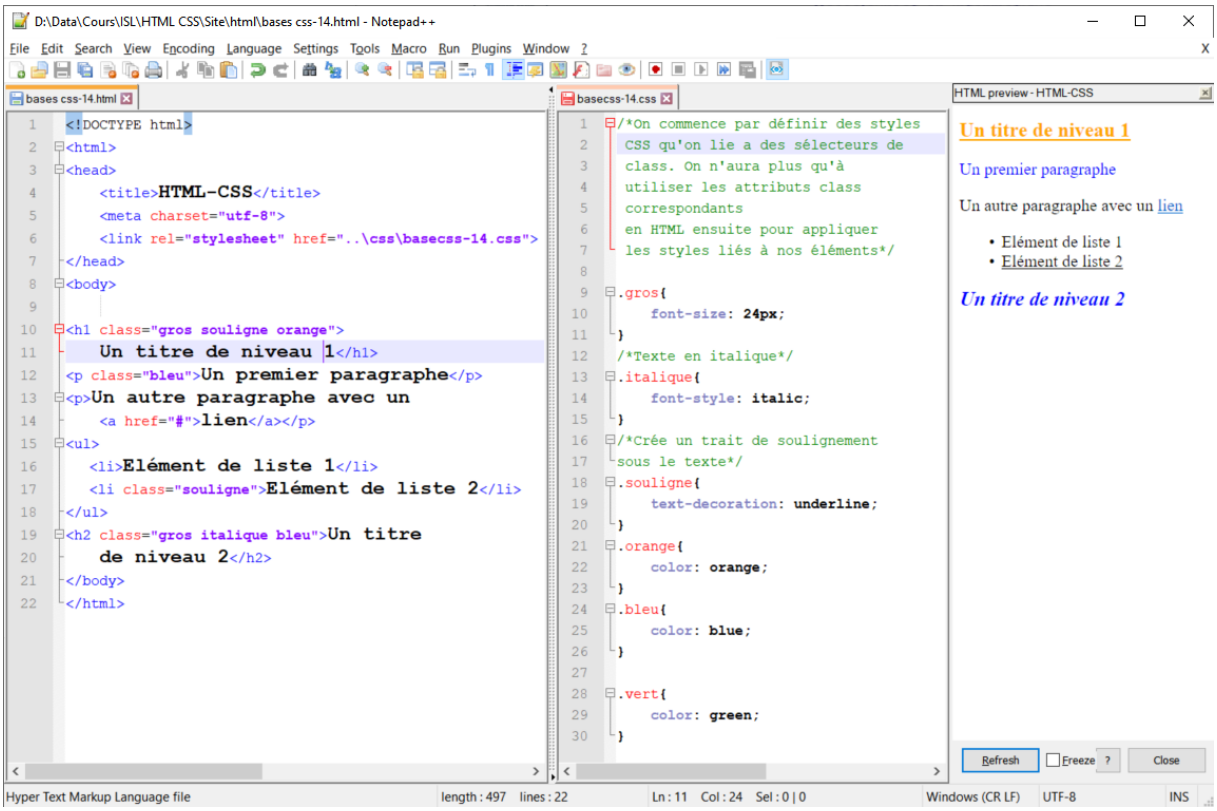
Cette notion de précision peut vous sembler un peu floue pour le moment car c'est le genre de notion qu'il est difficile de comprendre sans connaitre l'ensemble du langage. Pas d'inquiétude, tout cela va se préciser au fil des leçons et à chaque nouvelle notion que nous allons aborder.

## Attribuer plusieurs attributs class à un élément HTML

L'un des grands intérêts de l'attribut HTML class est qu'un même attribut (et donc les styles CSS liés) va pouvoir être partagé par différents éléments. Cela facilite grandement la gestion des styles de nos fichiers HTML et nous permet de gagner beaucoup de temps.

Réciproquement, un même élément HTML va tout à fait pouvoir recevoir différents attributs. Pour cela, il va suffire d'indiquer les différentes valeurs séparées par un espace. Ainsi, une très bonne pratique en CSS et pour la création d'un site va être de na pas surcharger un sélecteur .class avec de nombreux styles CSS mais au contraire d'utiliser de multiples sélecteurs .class qui se contenteront de définir chacun un comportement ou plusieurs propriétés d'un même « type ».

Fonctionner comme cela permet d'avoir un code beaucoup plus clair et d'avancer beaucoup plus vite dans la création d'un site. Regardez plutôt l'exemple ci-dessous pour bien comprendre cette utilisation des attributs class :




*Notez que faire ceci avec des attributs id n'aurait aucun sens puisque les styles des sélecteurs liés n'ont pas vocation à être partagés entre différents éléments (c'est-à-dire à être « réutilisés ») mais sont liés à un élément en particulier et il est donc logique ici de placer tous les styles voulus pour l'élément dans un seul id.*

# Ordre d'application (cascade) et héritage des règles en CSS

Dans cette nouvelle leçon, nous allons étudier et comprendre les mécanismes de cascade et d'héritage en CSS qui sont deux mécanismes fondamentaux de ce langage.

Comprendre comment fonctionne ces mécanismes va nous permettre de savoir quelle règle CSS va être appliquée à quel élément et pourquoi et ainsi de véritablement contrôler le résultat graphique de nos pages HTML.

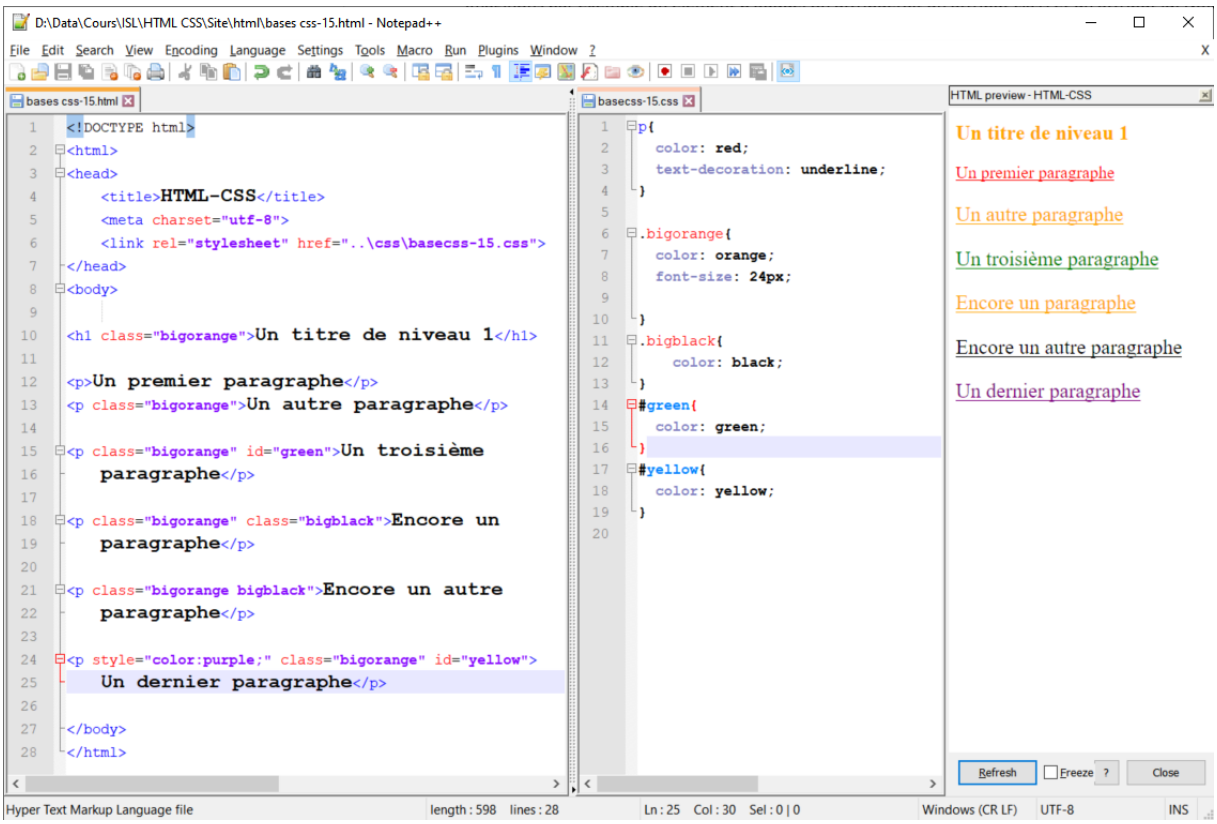
## Comprendre l'importance d'établir un ordre d'application des règles CSS : le problème des conflits

Pour comprendre les mécanismes fondamentaux de cascade et d'héritage en CSS, il faut avant tout comprendre ce qu'est un conflit CSS.

Parfois, plusieurs sélecteurs différents vont nous permettre d'appliquer des styles CSS à un même élément.

Imaginons par exemple un élément p auquel on attribuerait un attribut class et un attribut id. Nous allons pouvoir appliquer des styles CSS à cet élément de trois façons évidentes différentes :

- en utilisant un sélecteur élément ;
- en le ciblant via son attribut class ;
- en le ciblant via son attribut id.




```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="..\css\basecss-15.css">
</head>
<body>

<h1 class="bigorange">Un titre de niveau 1</h1>

<p>Un premier paragraphe</p>
<p class="bigorange">Un autre paragraphe</p>

<p class="bigorange" id="green">Un troisième
   paragraphe</p>

<p class="bigorange" class="bigblack">Encore un
   paragraphe</p>

<p class="bigorange bigblack">Encore un autre
   paragraphe</p>

<p style="color:purple;" class="bigorange" id="yellow">
   Un dernier paragraphe</p>

</body>
</html>
```

```
p{
   color: red;
   text-decoration: underline;
}

.bigorange{
   color: orange;
   font-size: 24px;

}
.bigblack{
    color: black;
}
#green{
   color: green;
}
#yellow{
   color: yellow;
}
```

Dans l'exemple ci-dessus, par exemple, nous avons une page HTML qui contient un titre de niveau 1 et trois paragraphes. Un de nos paragraphes possède un attribut class="bigorange" tandis qu'un autre possède à la fois un attribut class="bigorange" et un attribut id="green".

Enfin, un dernier paragraphe possède à la fois un attribut class="bigorange", un attribut id="yellow" et un attribut style dans lequel nous allons directement préciser un comportement pour la propriété color qui ne s'appliquera donc qu'à cet élément.

Du côté du CSS, on cible nos éléments HTML via quatre sélecteurs : un sélecteur éléments p, un sélecteur .bigorange et deux sélecteurs #green et #yellow. Certains de nos paragraphes vont donc être ciblés plusieurs fois avec plusieurs sélecteurs différents et recevoir les styles définis dans ces différents sélecteurs.

Ici, on voit que le sélecteur p est le seul sélecteur qui définit le comportement de la propriété text-decoration tandis que le sélecteur .bigorange est le seul qui définit le comportement de la propriété font-size. Il n'y aura donc pas de conflit sur ces deux propriétés puisqu'elles ne sont définies qu'une fois en CSS pour les mêmes éléments.

En revanche, on définit un comportement différent pour la propriété color au sein de chaque sélecteur. Dans ce cas-là, il va y avoir un conflit puisque le CSS va devoir déterminer quelle valeur de la propriété appliquer pour chaque élément ciblé avec plusieurs sélecteurs.

Pour comprendre comment le CSS va procéder dans ce cas, il faut avant tout bien se persuader que le CSS (comme tout autre langage web) repose sur un ensemble de règles. Les règles définissant

l'ordre de préférence d'application des propriétés définies dans différents sélecteurs sont contrôlées par un mécanisme qu'on appelle la cascade. Connaitre ces règles va nous permettre de prédire quel style sera appliqué dans telle ou telle situation.

## Le mécanisme de cascade CSS

Il n'est pas toujours simple de prédire quels styles CSS vont s'appliquer à quel élément pour la simple et bonne raison que le CSS peut être défini à des endroits différents (dans un élément style, dans la balise ouvrante d'un élément dans un attribut style ou dans un fichier CSS séparé) et qu'on va également pouvoir appliquer des styles à un élément en particulier en le ciblant via plusieurs sélecteurs CSS différents.

Il est donc essentiel de bien comprendre comment le CSS va fonctionner pour déterminer quels styles devront être appliqués à tel élément. L'ordre de préférence et d'application d'un style va dépendre de trois grands facteurs qui vont être :

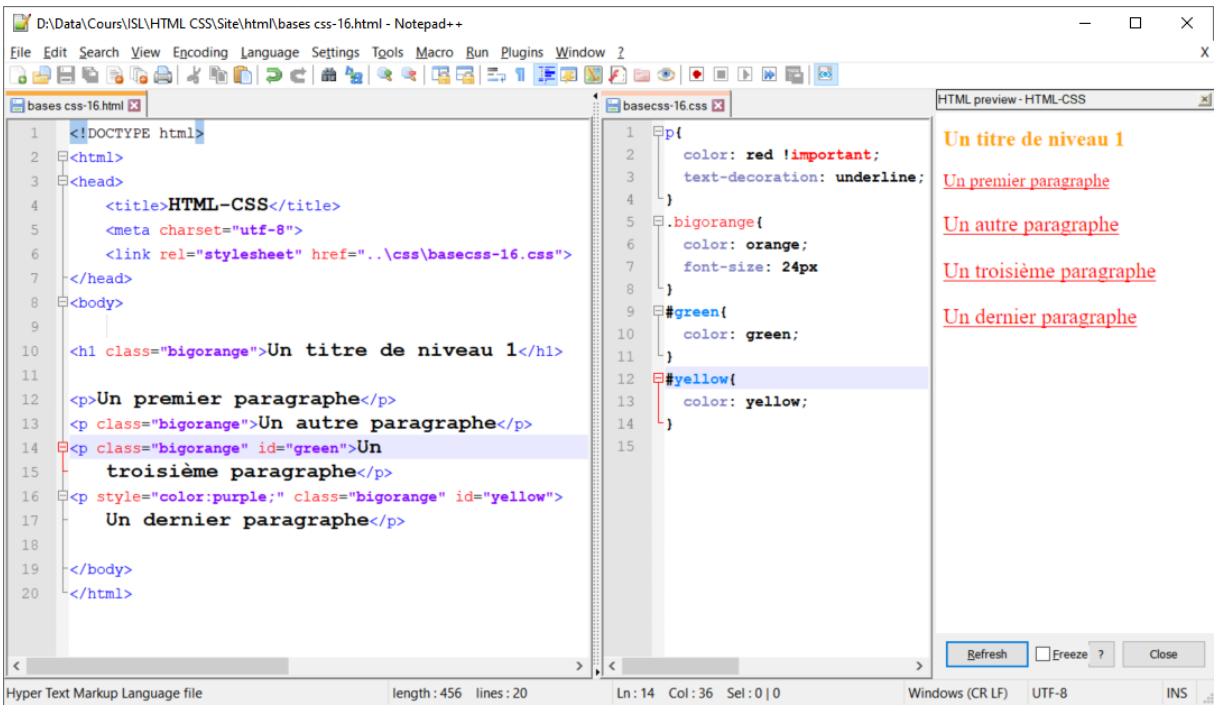
- La présence ou non du mot clef !important ;
- La précision du sélecteur ;
- L'ordre de déclaration dans le code ;

A noter que ces trois facteurs vont être analysés dans l'ordre donné et que le premier va primer sur le deuxième qui va primer sur le dernier : par exemple, si une règle utilise la syntaxe !important elle sera jugée comme prioritaire peu importe la précision du sélecteur ou sa place dans le code.

## Le mot clef !important

Le mot clef !important sert à forcer l'application d'une règle CSS. La règle en question sera alors considérée comme prioritaire sur toutes les autres déclarations et le style sera appliqué à l'élément concerné.

Nous allons placer ce mot clef à la fin d'une déclaration CSS lorsqu'on souhaite qu'un style s'applique absolument à un élément.




Comme vous pouvez le constater dans l'exemple ci-dessus, le fait d'ajouter !important dans la définition du comportement de la propriété color liée à notre sélecteur p fait que c'est cette définition qui s'appliquera par-dessus toutes les autres.

Ici, en particulier, vous pouvez voir que tous nos paragraphes sont rouges, même lorsque la propriété color a été définie différemment dans un sélecteur de class ou d'id et même lorsqu'un comportement différent a été précisé dans un attribut style dans la balise ouvrante d'un élément en particulier.

Le mot clef !important est donc extrêmement puissant en CSS et peut ainsi sembler très pratique et très utile aux yeux des débutants. Cependant, en pratique, nous essaierons tant que possible de nous en passer tout simplement car ce mot clef est une sorte de « joker » qui court-circuite toute la logique normale du CSS.

L'utiliser à outrance et lorsque ce n'est pas strictement nécessaire peut donc amener de nombreux problèmes par la suite comme par exemple des problèmes de styles définis autrement et qui ne s'appliqueraient pas car déjà définis avec !important ailleurs dans le code.

De manière générale, on préfèrera toujours aller dans le sens des langages et essayer de respecter et d'utiliser les normes qu'ils ont mis en place.

## Le degré de précision du sélecteur

Le deuxième critère déterminant dans l'application d'un style plutôt que d'un autre va être le degré de précision du sélecteur où le style a été défini une première fois par rapport aux autres degrés de précision des autres sélecteurs où le style a été à nouveau défini.

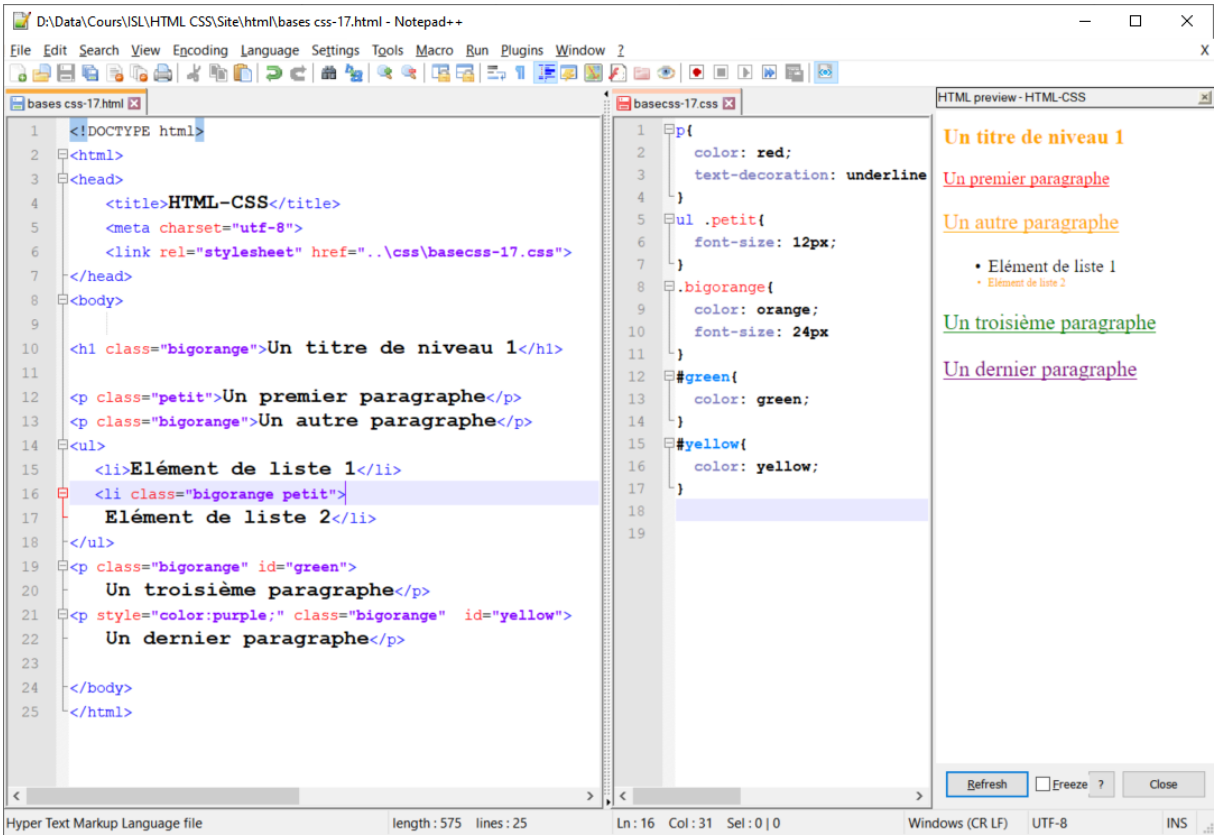
Le sélecteur le plus précis imposera ses styles aux sélecteurs moins précis en cas de conflit.

Pour rappel, la « précision » désigne ici le fait d'identifier de manière plus ou moins unique un élément. Les sélecteurs peuvent être rangés dans l'ordre suivant (du plus précis au moins précis) :

- Un style défini dans un attribut HTML style sera toujours le plus précis et notamment plus précis qu'un style défini avec un sélecteur CSS ;
- Le sélecteur #id va être le sélecteur le plus précis mais sera moins précis qu'un style défini dans un attribut HTML style ;
- Un sélecteur .class ou un autre sélecteur d'attribut* (*les autres sélecteurs d'attributs sont des sélecteurs complexes que nous étudierons plus tard) ou un sélecteur de pseudo-classe** (**nous verrons ce qu'est une pseudo-classe plus tard dans ce cours) sera moins précis qu'un sélecteur #id ;
- Un sélecteur d'élément ou de pseudo-élément*** (***nous étudierons les pseudo éléments plus tard dans ce cours) sera moins précis qu'un sélecteur d'attribut ou de pseudo-classe.

Si deux sélecteurs différents sont au même degré de précision, alors c'est le sélecteur le plus « complet » c'est-à-dire celui qui utilisera le plus de combinateurs qui sera jugé le plus précis.

Prenons immédiatement un exemple pour illustrer cela :



Ici, on voit que deux propriétés sont définies dans plusieurs sélecteurs qui servent à sélectionner le même élément : les propriétés color et font-size. Ce sont donc nos deux propriétés qui vont générer des conflits.

On commence par vérifier la présence du mot clef !important : il n'est défini nulle part ici. On passe donc au deuxième critère qui est le degré de précision.

On regarde déjà si des attributs style sont présents dans le code. C'est le cas pour notre dernier paragraphe qui possède un attribut style="color:purple". Comme la règle ne peut pas être appliquée plus précisément, on sait que ce paragraphe sera de couleur violette.

Ensuite, on s'intéresse à la présence d'attributs id. Notre troisième paragraphe possède un id="green" et le sélecteur correspondant définit la règle color : green. Ce paragraphe sera donc vert.

Ensuite, on regarde la présence de sélecteur .class ou de sélecteurs d'autres attributs ou de sélecteurs de pseudo-classes. Notre deuxième élément de liste possède deux attributs class : bigorange et petit et on va définir le comportement de la propriété font-size dans chacun des deux sélecteurs associés.

Ici, les deux sélecteurs sont des sélecteurs .class et possèdent donc le même degré de précision à priori. Il va donc falloir regarder si un sélecteur est plus complet que l'autre c'est-à-dire s'il utilise différents combinateurs pour le rendre plus précis ou pas. C'est le cas de notre sélecteur ul .petit qui va finalement nous servir à ne cibler que les éléments possédant un attribut class= "petit" contenus dans un élément ul.

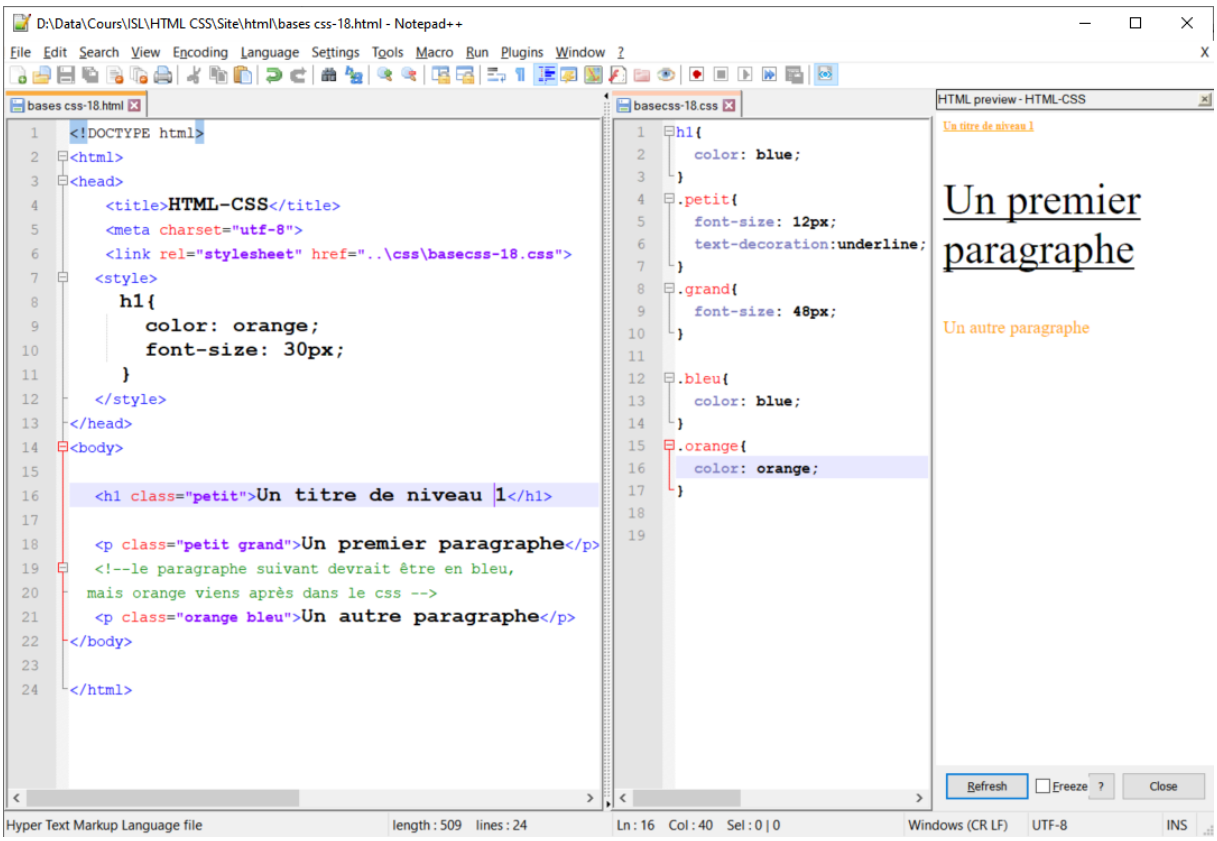
## L'ordre d'écriture des règles

Le troisième et dernier critère qui va nous permettre de définir quel style doit primer sur tel autre et doit donc être appliqué à un élément va tout simplement être l'ordre d'écriture d'une règle dans le code.

Ce critère va être utilisé dans le cas où plusieurs sélecteurs concurrents définissent le comportement d'une même propriété et ont la même importance et la même spécificité.

La règle ici est très simple : c'est la dernière déclaration dans le code qui primera sur des déclarations précédentes.

Regardez plutôt l'exemple suivant :




Ici, chacun de mes deux paragraphes possèdent deux attributs class qui vont à chaque fois définir le comportement d'une même propriété. Les sélecteurs CSS associés ont la même importance et le même degré de spécificité. Il va donc falloir regarder leur ordre d'écriture pour savoir quelles règles vont être appliquées.

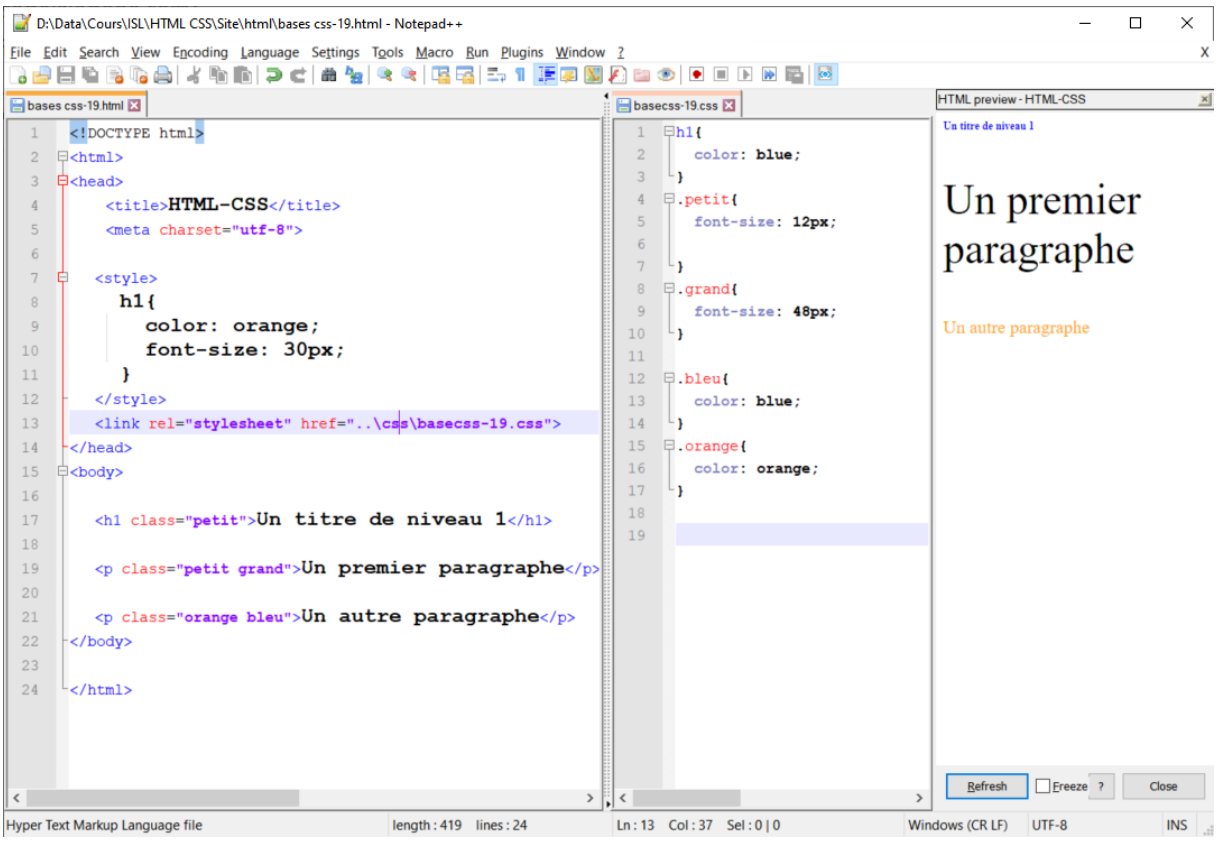
Ici, le sélecteur .grand apparait après le sélecteur .petit dans le code CSS. C'est donc la taille de texte définie dans .grand qui va être appliquée à notre premier paragraphe.

De même, le sélecteur .orange apparait après le sélecteur .bleu dans le code CSS. Le texte de notre deuxième paragraphe sera donc orange et non pas bleu.

Notez que c'est exactement la même règle d'ordre d'écriture des styles qui va s'appliquer, à sélecteur égal, pour déterminer si ce sont les styles définis dans un élément style ou si ce sont ceux définis dans un fichier CSS séparés qui vont s'appliquer.

Ici, notre titre h1 s'affiche en orange car nous avons précisé l'élément style après l'élément link qui fait appel à notre fichier CSS dans notre fichier HTML. Les styles définis dans l'élément style seront donc lus après ceux définis dans notre fichier CSS liés et seront donc appliqués dans le cas où plusieurs sélecteurs concurrents définissent le comportement d'une même propriété et ont la même importance et la même spécificité.

Pour vous en convaincre, échangeons la place des éléments link et style dans notre code HTML et observons le résultat sur notre code HTML :




Une convention en HTML va être de toujours préciser notre élément style après notre élément link dans le code pour ne pas s'embrouiller et c'est la raison pour laquelle on retient généralement qu'à sélecteur égal les styles définis dans l'élément style sont prioritaires sur ceux définis dans un fichier CSS séparé.

Notez qu'ici notre titre h1 va toujours avoir la taille définie dans le sélecteur .petit puisqu'un sélecteur d'attribut class est toujours plus précis qu'un sélecteur élément et que ce critère de précision passe avant le critère de l'ordre d'écriture des styles dans le code.

## L'héritage en CSS

La notion d'héritage est une autre notion fondamentale du CSS. Elle signifie que certains styles CSS appliqués à un élément vont être hérités par les éléments enfants de cet élément, c'est-à-dire par les éléments contenus dans cet élément.

Cette notion d'héritage est conditionnée par deux choses :

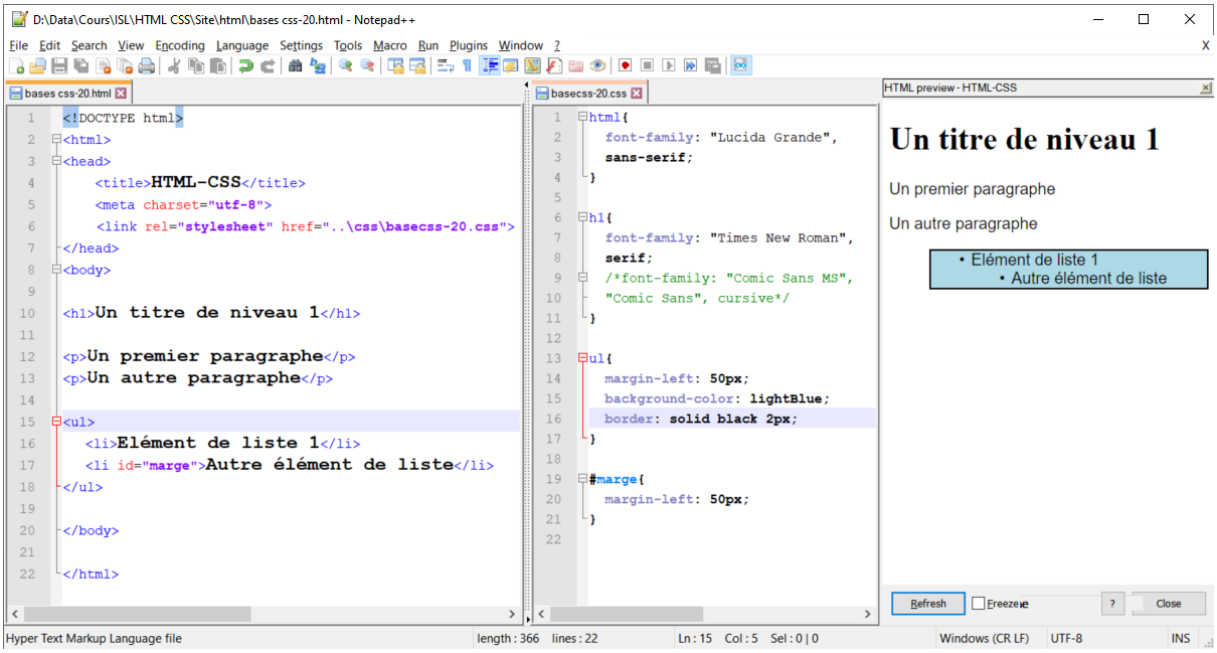
- Toutes les propriétés ne vont pas être héritées pour la simple et bonne raison que cela ne ferait aucun sens pour certaines de l'être ;

- Les éléments enfants n'hériteront des styles de leur parent que si il n'y a pas de conflit c'est-à-dire uniquement dans la situation où ces mêmes styles n'ont pas été redéfinis pour ces éléments enfants en CSS.

Pour savoir quelles propriétés vont pouvoir être héritées et quelles autres ne vont pas pouvoir l'être il va soit falloir faire preuve de logique (et bien connaitre le langage CSS), soit falloir apprendre par cœur pour chaque propriété si elle peut être héritée ou pas.

Les propriétés qui vont pouvoir être héritées sont en effet celles dont l'héritage fait du sens. Par exemple, la propriété font-family qui sert à définir un jeu de polices à utiliser pour du texte va pouvoir être hérité car il semble logique que l'on souhaite avoir la même police pour tous les textes de nos différents éléments par défaut.

En revanche, les propriétés liées aux marges par exemple ou plus généralement les propriétés de mise en page et de positionnement des éléments ne vont pas pouvoir être héritées car cela ne ferait aucun sens d'un point de vue design de rajouter une marge d'une taille définie pour chaque élément enfant.




Dans l'exemple ci-dessus, je définis un jeu de police avec la propriété font-family dans mon sélecteur html. Comme tous les éléments d'une page HTML sont des enfants de cet élément (ils sont contenus dans l'élément html) et comme la propriété font-family peut être héritée, tous les textes de ma page utiliseront la police d'écriture définie dans cette propriété sauf si une autre police est définie de manière plus précise avec un sélecteur plus précis comme c'est le cas pour mon titre h1 ici.

J'attribue ensuite une marge extérieure gauche égale à 50px à mon élément ul représentant ma liste. Ma liste sera donc décalée de 50px par rapport au bord gauche de son élément parent qui est ici l'élément body qui représente la page. Cependant, comme la propriété margin ne peut pas être héritée, les éléments de liste de vont pas hériter de ce même margin-left : 50px par défaut.

Ici, vous devez bien comprendre que la marge se calcule par rapport au début de l'élément parent. La liste entière est décalée de 50px par rapport à l'élément body mais les éléments de liste ne sont pas décalés par défaut de 50px par rapport à l'élément ul qui est leur élément parent. Pour bien illustrer cela, j'ai ajouté manuellement un margin-left : 50px au deuxième élément de liste afin de vous prouver que le premier élément de liste n'a pas hérité de la propriété margin appliquée à son élément parent ul.

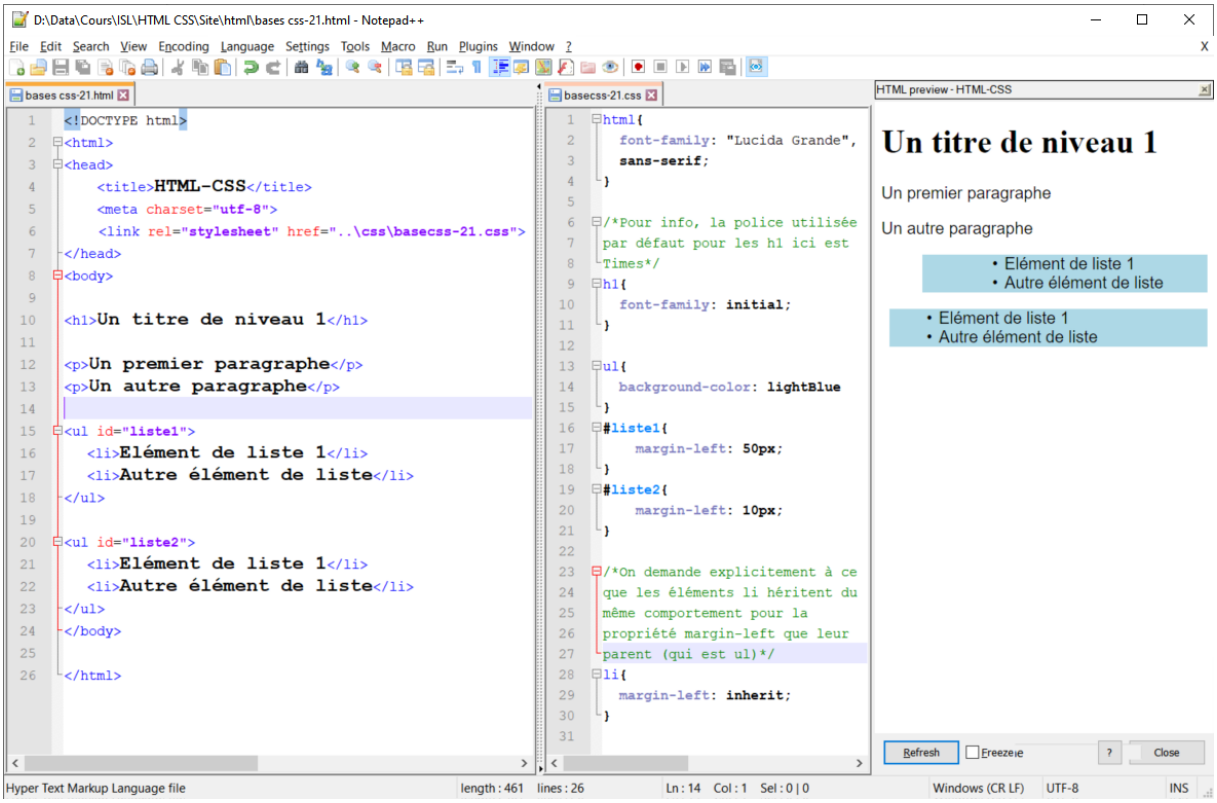
Notez que le CSS nous laisse toutefois la possibilité de « forcer » un héritage pour des propriétés non héritées par défaut ou plus exactement la possibilité de définir des comportements d'héritage pour chaque propriété définie dans chaque sélecteur.

Pour faire cela, nous allons pouvoir utiliser quatre valeurs qui vont fonctionner avec toutes les propriétés CSS (elles sont dites universelles) et qui vont nous permettre d'indiquer que telle propriété définie dans tel sélecteur doit avoir le même comportement que celle définie pour l'élément parent ou pas.

Ces valeurs sont les suivantes :

| Valeur | Signification |
|---|---|
| inherit | Sert à indiquer que la valeur de propriété appliquée à l'élément sélectionné est la même que celle de l'élément parent |
| initial | Sert à indiquer que la valeur de propriété appliquée à l'élément sélectionné est la même que celle définie pour cet élément dans la feuille de style par défaut du navigateur |
| unset | Permet de réinitialiser la propriété à sa valeur naturelle, ce qui signifie que si la propriété est naturellement héritée elle se comporte comme si on avait donné la valeur inherit. Dans le cas contraire, son comportement sera le même que si on lui avait donné la valeur initial |
| revert | Permet la propriété à la valeur qu'elle aurait eue si aucun style ne lui avait été appliqué. La valeur de la propriété va donc être fixée à celle de la feuille de style de l'utilisateur si elle est définie ou sera récupérée dans la feuille de style par défaut de l'agent utilisateur |

En pratique, la valeur la plus utilisée parmi ces quatre va être inherit. Notez également que le support pour la valeur revert n'est pas encore acquis pour la plupart des navigateurs. Je n'ai évoqué cette valeur ici que par souci d'exhaustivité mais vous déconseille de l'utiliser pour le moment. Pour cette raison, je ne l'évoquerai plus dans la suite de ce cours.




Ici, on définit un h1{font-family: initial;} en CSS. Ainsi, c'est la valeur de font-family définie pour cet élément dans la feuille de style par défaut du navigateur qui va être appliquée. En l'occurrence, dans mon cas, cela va être la valeur Times.

Ensuite, on demande explicitement à ce que les éléments de liste li héritent de la valeur donnée à la propriété margin-left à leur parent. Pour notre première liste, on définit margin-left: 50px. Les éléments li vont donc également posséder une marge extérieure gauche de 50px par rapport à la liste en soi qui est leur élément parent.

Pour notre deuxième liste, en revanche, on a défini une marge gauche de 10px seulement. Les éléments de liste vont donc utiliser cette même valeur pour leur propriété margin-left et être décalés de 10px par rapport à la liste en soi.

## Conclusion sur les mécanismes de cascade et d'héritage en CSS

Les mécanismes de cascade et d'héritage en CSS vont permettre de définir via un ensemble de règles quels styles vont être appliqués à quel élément en cas de conflit.

Ces mécanismes vont en pratique très souvent entrer en jeu. En effet, la plupart des sites sont aujourd'hui complexes et vont utiliser plusieurs feuilles de styles (fichiers CSS) différentes qui vont définir de nombreuses règles à appliquer à chaque élément.

Comprendre comment ces mécanismes fonctionnent et connaitre ces règles est essentiel et fondamental puisque cela va nous permettre de toujours obtenir le résultat visuel espéré.

Notez que la cascade et l'héritage sont le cœur même du CSS et sont en grande partie sa puissance puisque ces mécanismes vont nous permettre d'un côté de pouvoir « surcharger » des styles en utilisant des sélecteurs plus ou moins précis et de l'autre côté de transmettre des styles d'un élément parent à ces enfants et donc nous éviter de définir tous les styles voulus pour chaque élément.

# Les éléments HTML div et span (conteneurs génériques)

Dans cette nouvelle leçon, nous allons nous intéresser à deux éléments HTML très spéciaux qui sont les éléments div et span.

Ces éléments sont très particuliers puisqu'ils ne servent pas à préciser la nature d'un contenu mais vont simplement nous servir de conteneurs génériques en HTML.

Nous allons ici comprendre l'intérêt de ces deux éléments et en particulier leur intérêt pour l'application de styles CSS et les cas d'utilisation de ces éléments.

## Le HTML et la valeur sémantique des éléments

Il est toujours bon de commencer par rappeler le rôle du HTML : le HTML a pour but de structurer du contenu et de lui donner du sens.

Les éléments HTML vont nous servir à marquer les différents contenus et donc à indiquer aux navigateurs et moteurs de recherche de quoi est composé une page. On va ainsi pouvoir dire grâce au HTML que tel contenu doit être considéré et traité comme un paragraphe, que tel autre contenu est un titre, que tel texte est plus important qu'un autre, que ceci est une liste, que cet objet est une image, etc.

A ce titre, les éléments HTML div et span sont très spéciaux puisque ce sont deux éléments HTML qui ne possèdent aucune valeur sémantique, c'est-à-dire qu'ils ne servent pas à préciser la nature d'un contenu.

Ces deux éléments sont en effet des conteneurs génériques qui ont été créés pour nous permettre d'ordonner nos pages plus simplement ensuite en CSS.

## Quels usages pour les éléments div et span ?

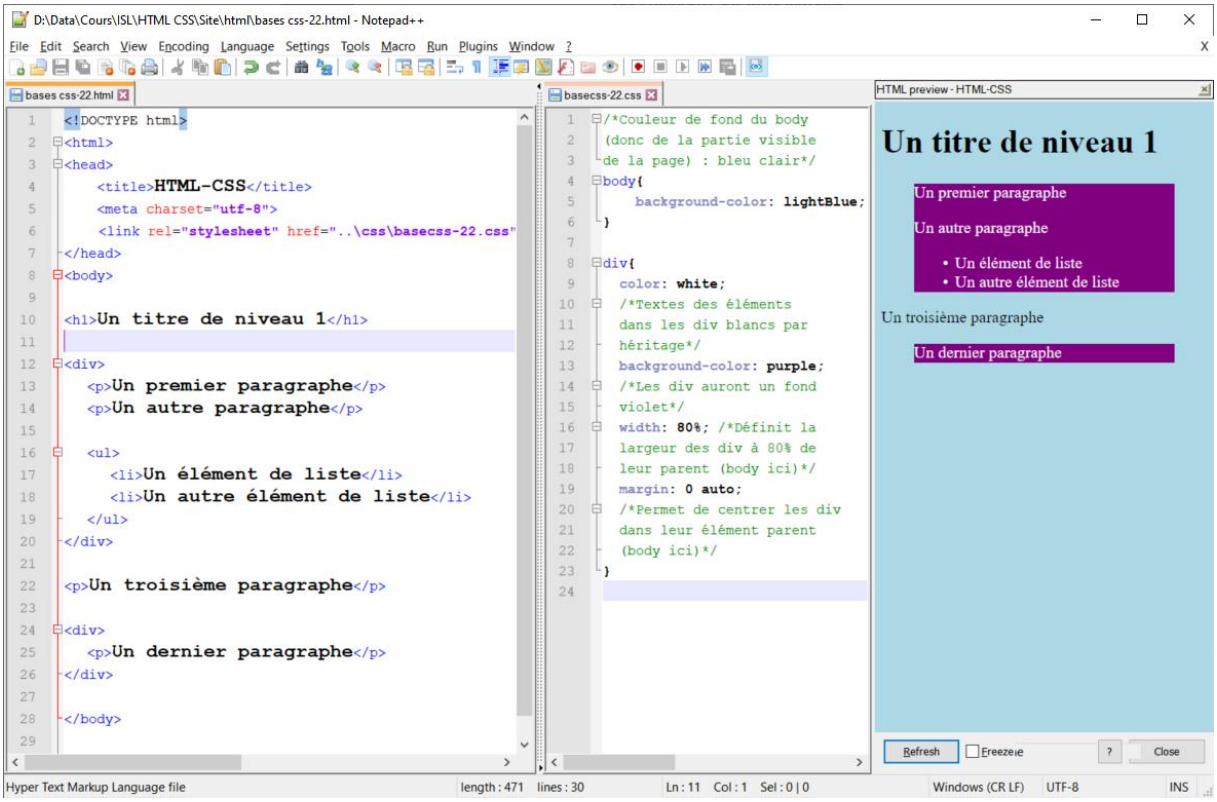
Le fait que les éléments div et span ne possèdent aucune valeur sémantique peut faire penser qu'ils vont à l'encontre même du rôle du HTML. C'est tout à fait vrai en soi, et c'est la raison pour laquelle on essaiera idéalement de n'utiliser ces éléments qu'en dernier recours et si nous n'avons aucun autre choix crédible.

Les éléments HTML div et span ont été créés principalement pour simplifier la mise en page de nos pages HTML en CSS c'est-à-dire pour simplifier l'application de certains styles CSS.

## Exemple d'utilisation de l'élément div

Nous allons utiliser l'élément div comme conteneur pour différents éléments afin de pouvoir ensuite facilement appliquer les mêmes styles CSS à tous les éléments contenus dans notre div par héritage ou pour les mettre en forme en appliquant un style spécifique au div.

Ici, le terme de « conteneur » est l'équivalent du terme « parent » : nous allons simplement placer nos différents éléments à l'intérieur de nos balises  et  puis appliquer les styles CSS directement au div.




On peut ici penser qu'on peut arriver au même résultat en utilisant plusieurs attributs class possédant la même valeur pour les différents éléments. Ce n'est pas tout à fait vrai.

Tout d'abord, ce n'est pas normalement exactement le rôle de base des attributs class : les sélecteurs .class sont censés être liés à un style CSS particulier et chaque élément doit pouvoir utiliser la class adaptée pour appliquer ce style. Ici, nous utilisons plutôt le sélecteur .class de manière exclusive en définissant de nombreux styles pour un groupe d'éléments en particulier. La logique de code est donc inversée.

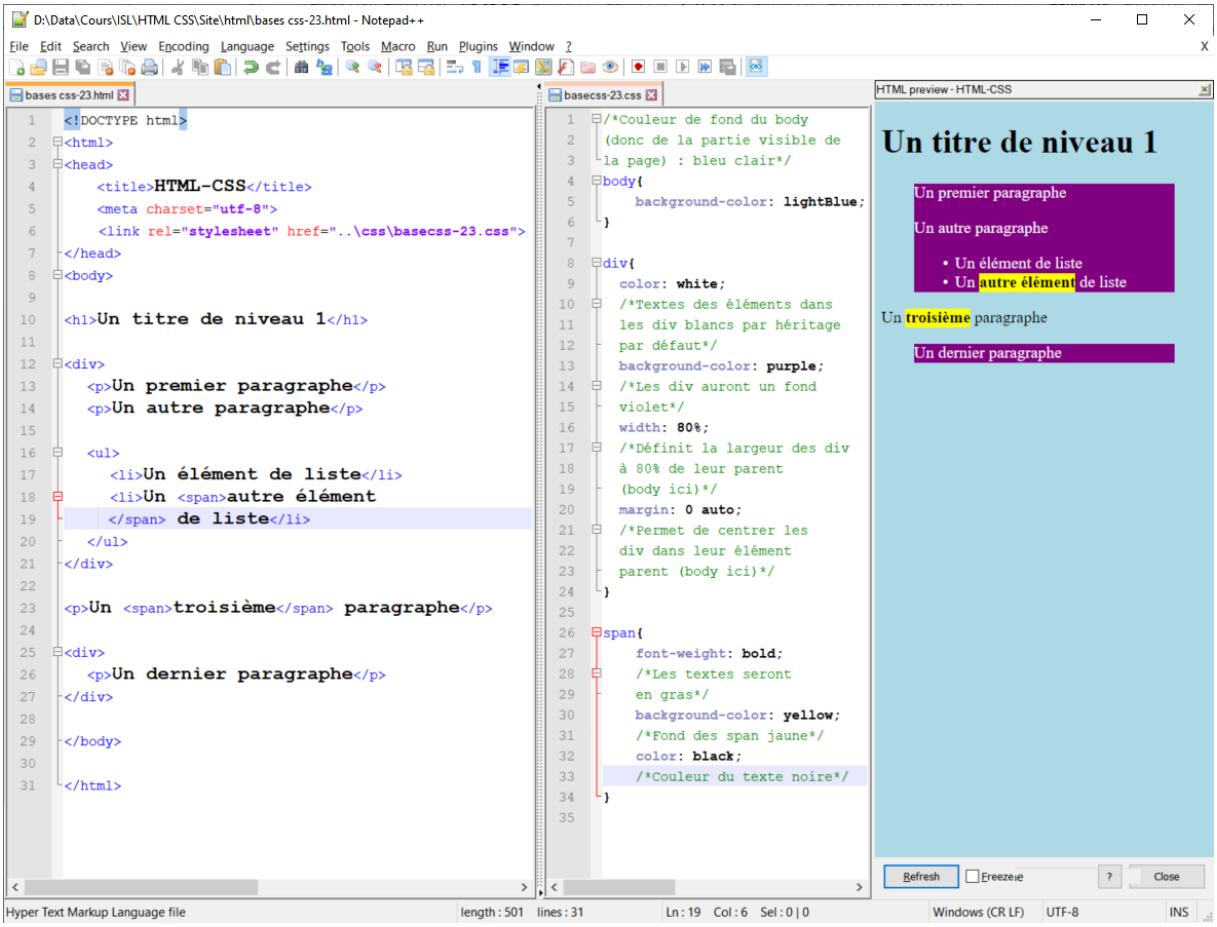
En dehors de cette considération sur le rôle des class, il est beaucoup plus simple et rapide dans le cas présent d'utiliser un div que de renseigner un attribut class à chaque fois.

De plus, dans certaines situations, nous allons vouloir pour des raisons de mise en page appliquer des styles spécifiquement au conteneur et nos pas à chaque élément contenu comme par exemple des marges externes.

## Exemple d'utilisation de l'élément span

L'élément span va lui servir de conteneur à un autre niveau : il va servir de conteneur interne à un élément plutôt que de conteneur pour plusieurs éléments.

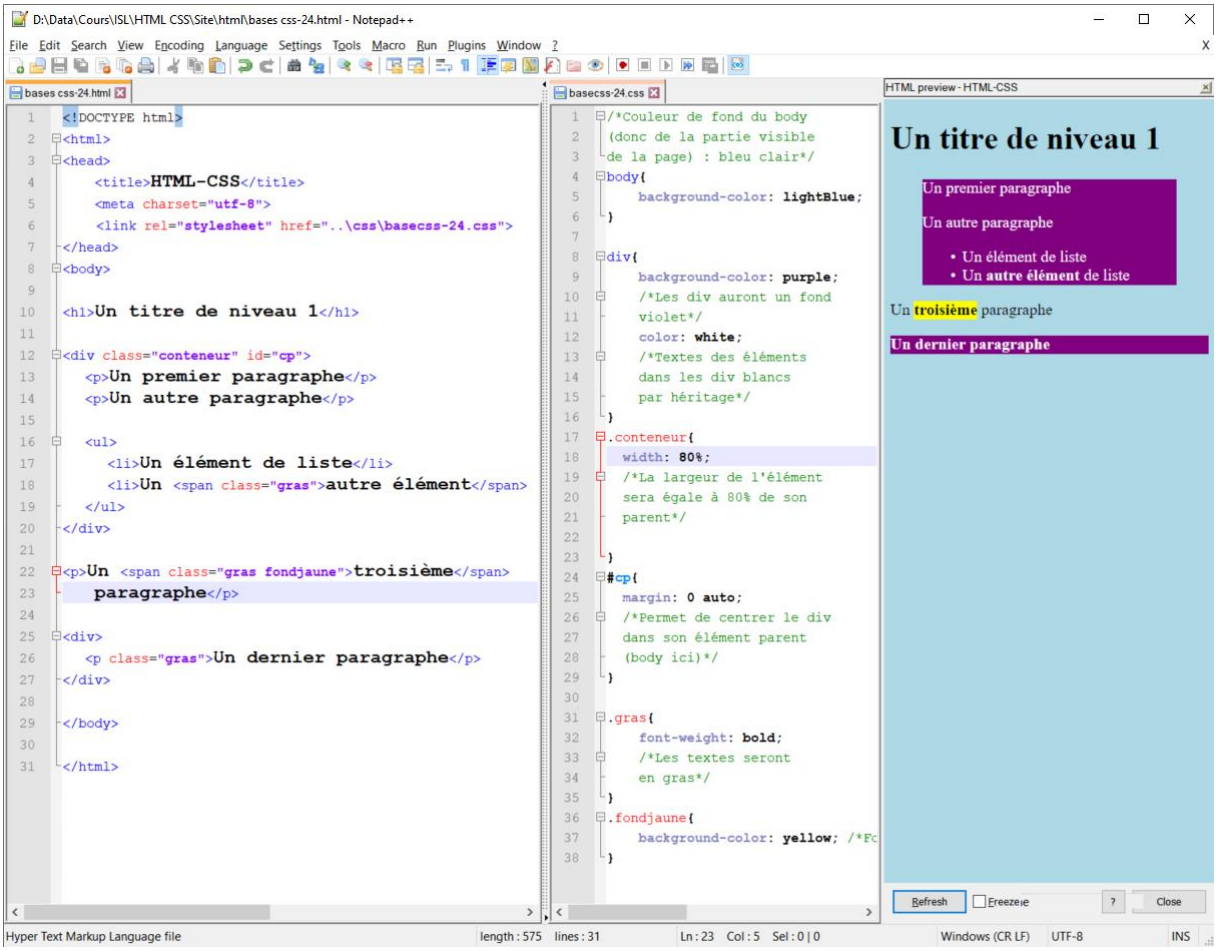
On va par exemple pouvoir placer une certaine partie du texte d'un titre ou d'un paragraphe dans un élément span pour ensuite pouvoir lui appliquer un style CSS particulier, chose qu'il nous était impossible de faire jusqu'à présent.




## Les éléments div et span et les attribut class et id

Les attributs class et id sont des attributs universels ce qui signifie qu'on va pouvoir les utiliser avec n'importe quel élément HTML, et notamment avec les éléments div et span.

En pratique, il va être très courant de préciser des attributs class et id pour nos éléments div et span pour pouvoir appliquer des styles à un div (ou span) ou à un groupe d'éléments div ou span) définis.b




Avec ce qu'on a appris dans la dernière leçon, vous devriez être capable de comprendre les styles appliqués ici en vous concentrant. Je vous laisse donc essayer, ça vous fera un bon exercice !

## Quelles différences entre les éléments div et span et quand utiliser l'un plutôt que l'autre ?

La grande différence entre les éléments div et span va concerner ce qu'ils vont pouvoir contenir : un élément div va pouvoir conteneur plusieurs éléments et va donc nous servir de conteneurs d'éléments tandis que l'élément span va nous servir de conteneur pour une partie du contenu d'un élément et va donc être utilisé à l'intérieur d'un élément.

Cette différence est due au fait que les éléments div et span sont de niveau ou au « type » différents : l'élément div est un élément de niveau block tandis que l'élément span est un élément de niveau inline.

Nous découvrirons plus tard dans ce cours ce que ces différents « niveaux » ou « types » d'éléments signifient et les différences entre eux.

# Les niveaux ou « types » d'éléments HTML block et inline

Nous avons pour le moment défini et étudié les grands mécanismes de fonctionnement du CSS tout en présentant certaines propriétés CSS impactant l'aspect visuel des éléments.

Pour aller plus loin dans notre étude du CSS, nous allons devoir maintenant comprendre comment est définie la place prise par chaque élément dans une page.

## Les niveaux ou « types » d'éléments HTML

De manière schématique, on peut considérer qu'Il existe deux grands types d'affichage principaux pour les éléments HTML : un élément HTML va pouvoir être soit de niveau (ou de type) block, soit de niveau (ou de type) inline.

Ces types d'affichage vont définir la façon dont les éléments vont se comporter dans une page par rapport aux autres et la place qu'ils vont prendre dans la page.

Connaître le type d'affichage d'un élément HTML va donc être essentiel pour créer et mettre en forme nos pages web car les éléments de type block et ceux de type inline vont se comporter de façon radicalement différente dans une page et certaines propriétés CSS vont avoir des comportements différents selon le type d'affichage d'un élément.

## Comprendre comment est défini le type d'affichage d'un élément HTML

Le type d'affichage d'un élément va toujours être défini en CSS par la propriété display. Si cette propriété n'est pas explicitement renseignée pour un élément, c'est la valeur par défaut de display qui va être appliquée à l'élément c'est-à-dire display: inline.

Ainsi, par défaut, on peut dire que tout élément HTML va posséder un type d'affichage inline (nous allons voir par la suite ce que signifie ce type d'affichage).

Cependant, rappelez-vous que chaque navigateur possède une feuille de styles (c'est-à-dire un fichier CSS) qui sera appliquée par défaut pour les différents éléments dont nous ne précisons pas le comportement dans nos propres feuilles de style.

La plupart des navigateurs possèdent aujourd'hui des feuilles de style similaires notamment pour la définition des styles basiques et c'est une très bonne chose pour nous développeur puisque cela va nous éviter d'avoir à définir le comportement de chaque propriété pour chaque élément de notre page.

Parmi ces styles par défaut appliqués par n'importe quel navigateur se trouve la définition du type d'affichage ou du display pour chaque élément.

Aujourd'hui, la plupart des navigateurs suivent les recommandations du W3C (l'organisme en charge de l'évolution et des standards des langages Web). Ce W3C spécifie pour chaque élément HTML quelle devrait être la valeur de son display.

Attention cependant : encore une fois, ce ne sont que des recommandations et chaque navigateur est libre de ne pas en tenir compte et de définir une autre valeur de display pour chaque élément !

C'est la raison pour laquelle il reste important de définir nous-mêmes la plupart des styles CSS impactant et qui pourraient être définis différemment par défaut par différents navigateurs.

Le W3C va donc indiquer quel devrait être le type d'affichage d'un élément par défaut et l'immense majorité des navigateurs va appliquer ces recommandations ce qui fait que l'un des display suivants à la plupart des éléments HTML en fonction de l'élément :

- display : block : affichage sous forme d'un bloc ;
- display : inline : affichage en ligne ;
- display : none : l'élément n'est pas affiché.

Notez que le W3C préconise d'autres types d'affichage pour certains éléments HTML particuliers. Les deux autres valeurs de display généralement respectées et appliquées par les navigateurs sont :

- display : list-item va être appliquée par défaut pour les éléments de liste li. L'affichage se fait sous forme de bloc mais une boite avec un marqueur est également générée ;
- display : table va être appliquée par défaut pour les éléments de tableau table. L'affichage se fait sous forme de bloc.

Certains navigateurs dans certains cas très particuliers peuvent également utiliser la valeur inline-block pour afficher certains éléments.

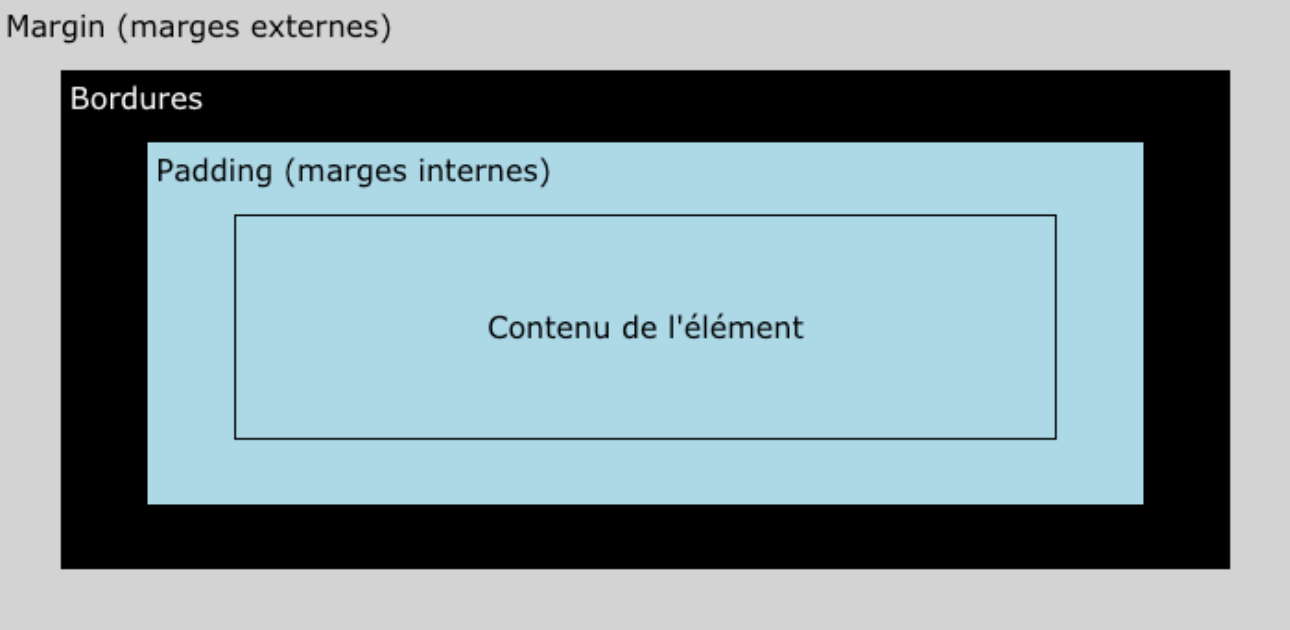
Ici, vous devez bien comprendre que ces valeurs ne sont que les valeurs préconisées par le W3C. Rien ne nous empêche de définir un type d'affichage différent de celui préconisé pour un élément en utilisant la propriété display avec la valeur souhaitée. Cela va pouvoir être utile pour aider à la mise en page de certains éléments.

***Bon à savoir** : Jusqu'à récemment (jusqu'au HTML 4.1), le W3C utilisait la simple distinction « bloc-level elements » vs « inline-level elements » (éléments de type block vs éléments de type inline) pour catégoriser les éléments HTML. Cependant, l'évolution des langages HTML et CSS et de leur complexité a amené le W3C à repenser la façon dont les éléments devaient être catégorisés. Aujourd'hui, donc, les éléments sont classés selon des catégories ou des modèles de contenus (« content categories » ou « content models »). Nous reparlerons de ces concepts avancés plus tard.*

## Rapide introduction au modèle des boites

Le modèle des boites (que nous étudierons plus tard dans ce cours) nous dit que tout élément HTML peut être représenté sous forme d'une boite rectangulaire. C'est une représentation qu'il vous faut connaitre et qu'il vous faudra comprendre.

Cette boite rectangulaire représentant l'élément contient d'autres boites (une qui contient le contenu, une autre qui contient en plus les marges intérieures, etc.) et ces différentes boites vont se comporter de manière différente selon le type d'affichage qui va lui être attribué, c'est-à-dire selon la valeur donnée à la propriété display pour cet élément.




Cependant, retenez bien ici que quelle que soit la valeur donnée à la propriété display, l'élément va toujours pouvoir être représenté sous forme d'une boite. Cela peut vous sembler flou pour le moment, mais intégrer cela va beaucoup vous aider pour bien comprendre comment créer des mises en page efficaces en CSS.

## Les éléments de type inline

Par simplicité, on appellera « élément de type inline » (ou « inline level element » en anglais) un élément auquel a été appliqué un display: inline.

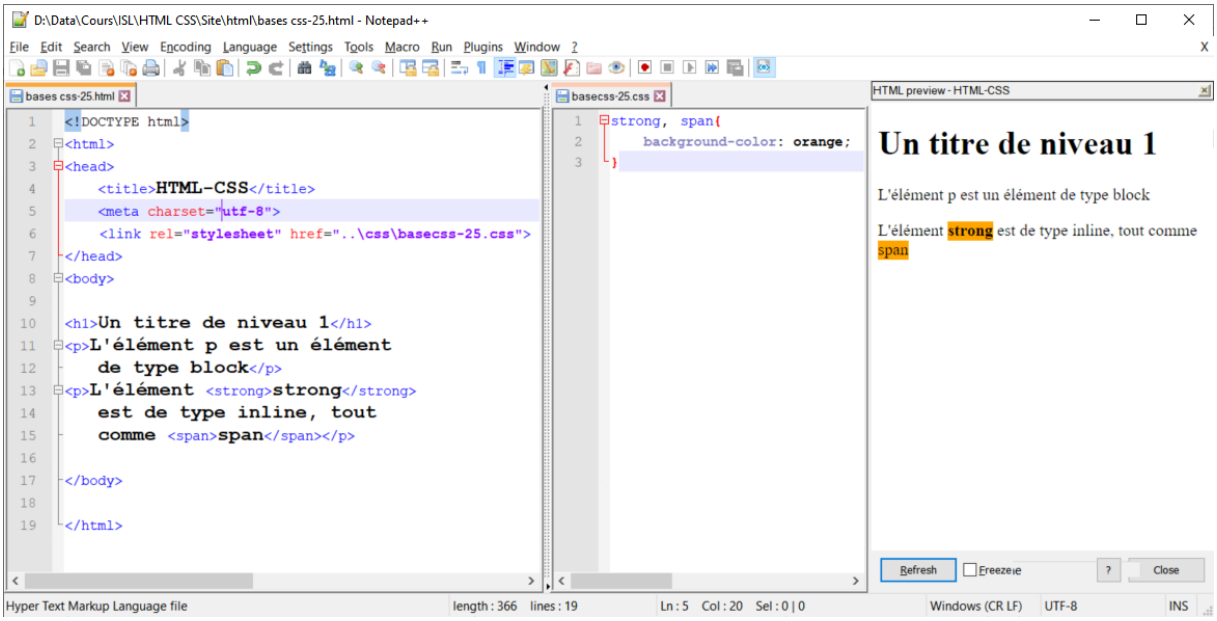
Les éléments de type inline vont posséder les caractéristiques suivantes qui vont les différencier des éléments de type block :

- Un élément de type inline ne va occuper que la largeur nécessaire à l'affichage de son contenu par défaut ;
- Les éléments de type inline vont venir essayer de se placer en ligne, c'est-à-dire à côté (sur la même ligne) que l'élément qui les précède dans le code HTML ;
- Un élément de type inline peut contenir d'autres éléments de type inline mais ne devrait pas contenir d'éléments de type block.

De plus, notez qu'on ne va pas par défaut pouvoir appliquer de propriété width ou height à un élément de type inline puisque la caractéristique principale de ce type d'éléments est de n'occuper que la place nécessaire à l'affichage de leur contenu.

Les éléments HTML dont le type d'affichage recommandé par le W3C est le type inline les plus courants sont les suivants :

- Les éléments de distinction d'importance du contenu em et strong ;
- L'élément span ;
- L'élément de liens a ;
- L'élément button ;
- Les éléments de formulaire input, label, textarea et de liste de choix select ;
- L'élément d'insertion d'images img (cas intéressant et souvent source de confusions car on va pouvoir passer une largeur et une hauteur à l'image à afficher en soi qui va « remplacer » l'élément img lors de l'affichage, mais il n'empêche que l'élément img est bien inline en soi);
- Les éléments code, script, etc.




Dans l'exemple ci-dessus, j'ai rajouté une couleur de fond aux éléments inline afin que vous puissiez bien voir l'espace qu'ils prennent dans la page.

## Les éléments de type block

Par simplicité, on appellera « élément de type block » (ou « block level element » en anglais) un élément auquel on va appliquer un display: block.

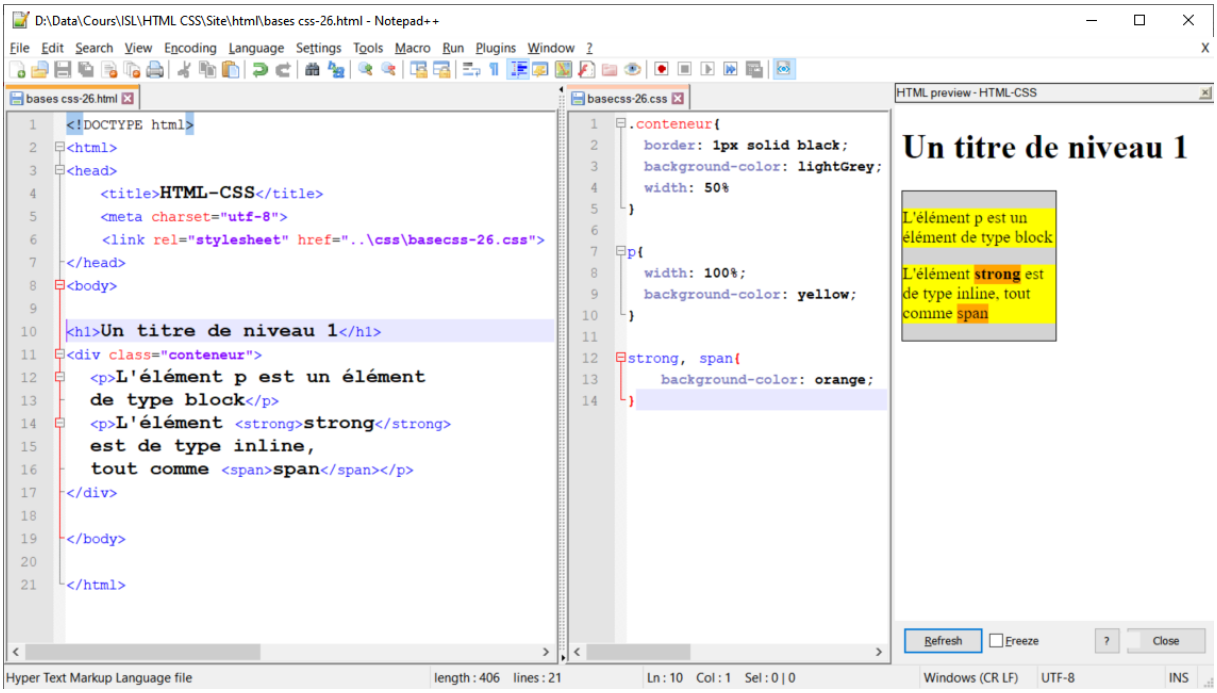
Les éléments de type block vont posséder les caractéristiques de disposition suivantes :

- Un élément de type block va toujours prendre toute la largeur disponible au sein de son élément parent (ou élément conteneur) ;

- Un élément de type block va toujours « aller à la ligne » (créer un saut de ligne avant et après l'élément), c'est-à-dire occuper une nouvelle ligne dans une page et ne jamais se positionner à côté d'un autre élément par défaut ;
- Un élément de type block peut contenir d'autres éléments de type block ou de type inline.

Les éléments HTML dont le type d'affichage recommandé par le W3C est le type block les plus communs sont les suivants :

- L'élément body, cas particulier mais qui est concrètement considéré comme un élément block ;
- L'élément de division du contenu div ;
- Les paragraphes p et titres h1, h2, h3, h4, h5, h6 ;
- Les éléments structurants article, aside, header, footer, nav et section ;
- Les listes ul, ol, dl et éléments de listes de définition dt et dd ;
- L'élément de définition de tableaux table ;
- L'élément de définition de formulaires form et l'élément fieldset ;
- Les éléments figure et figcaption ;
- Les éléments canvas, video, etc.



**Bon à savoir** : Les éléments HTML comme video et img décrits ci-dessus comme étant respectivement de types block et inline sont des éléments HTML très particuliers : ils sont dans la catégorie des éléments « void » (j'utilise ici le mot anglais car utiliser « vide » porterait à confusion) et ce sont également ce qu'on appelle des éléments HTML remplacés. Un élément void est un élément qui ne peut pas posséder de contenu qui lui soit propre et un élément remplacé est un élément qui fait appel à du contenu extérieur (qui va être « remplacé » par un contenu extérieur) possédant déjà des dimensions propres. Notez que la plupart des éléments remplacés sont également des éléments void.

## Les autres valeurs de la propriété display

Nous allons passer de nombreuses autres valeurs à la propriété display en plus des valeurs inline, block et none comme par exemple inline-block, table, list-item, flex, etc.

Nous aurons l'occasion de reparler des différentes valeurs de la propriété display dans la leçon qui lui est dédiée plus tard dans ce cours.

Pour le moment, retenez simplement ici que toutes ces « sous » valeurs d'affichage vont toujours reposer sur un affichage pour l'élément en soi block ou inline auquel vont pouvoir s'ajouter différentes règles, contraintes ou variations.

Je vous demande donc pour l'instant de considérer que tous les éléments sont soit de type block, soit de type inline et de bien retenir les différences entre ces deux types d'affichage.

# Notations complètes « long hand » et raccourcies « short hand » CSS

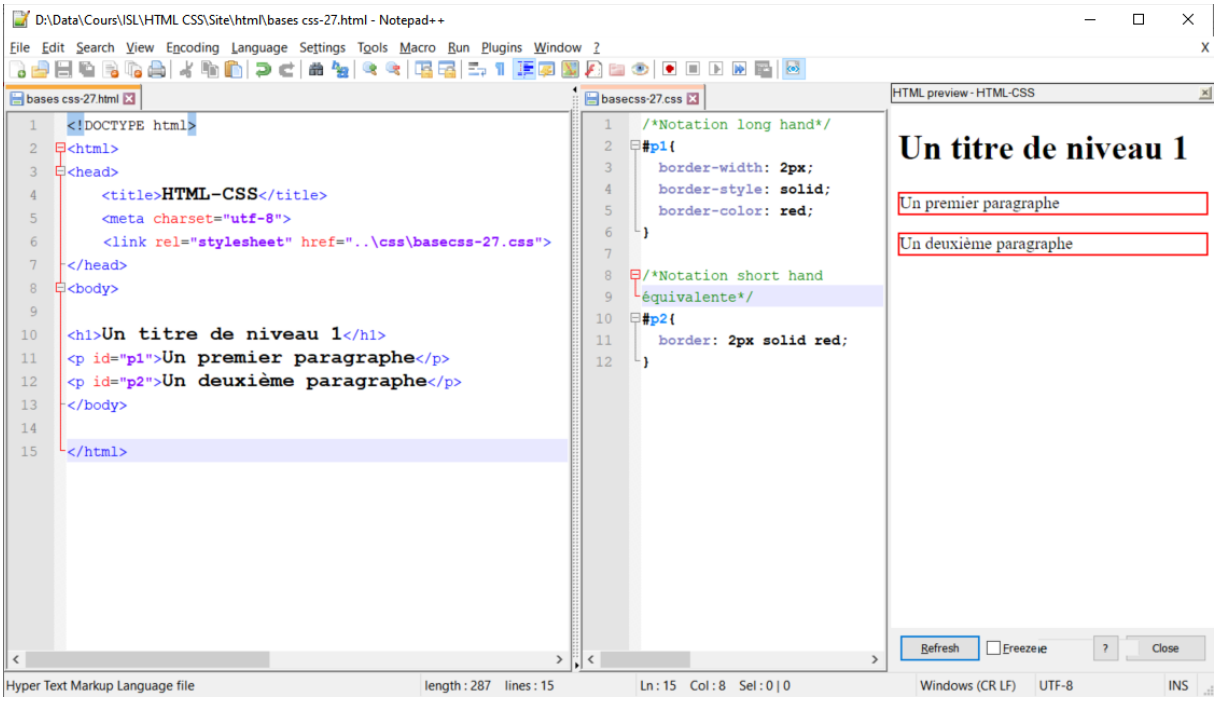
Dans cette nouvelle leçon, nous allons définir ce que sont les notations CSS raccourcies ou « short hand » en anglais et voir à quel moment il est intéressant de les utiliser par rapport aux notations complètes ou « long hand ».

## Définition d'une notation CSS raccourcie ou notation « short hand »

Une notation raccourcie ou notation « short hand » ou encore « propriété raccourcie / short hand » correspond à une propriété à laquelle on va pouvoir passer les valeurs d'un ensemble d'autres propriétés et qui va donc nous permettre de définir de valeur de plusieurs propriétés d'un coup.

Nous avons déjà été amené à en rencontrer certaines dans ce cours comme par exemple la propriété border qui correspond en fait à la notation raccourcie des propriétés border-width, border-style et border-color et qui nous permet ainsi de définir d'un coup les valeurs pour ces trois propriétés pour un élément.
Ex : border:1px solid black;

De manière générale, il va souvent être équivalent de préciser le comportement d'un aspect d'un élément HTML en utilisant une notation raccourcie ou en utilisant les propriétés long hand.




Cependant, les notations short hand possèdent certains avantages sur les notations long hand mais également certaines limites dans certaines situations que nous allons voir dans la suite de cette leçon.

## L'ordre de déclaration des valeurs des propriétés short hand

Une propriété short-hand est une propriété à laquelle on va pouvoir passer les valeurs de plusieurs autres propriétés.

L'ordre de déclaration des valeurs ne va compter que dans le cas où la notation raccourcie va accepter plusieurs valeurs d'un type similaire et donc dans le cas où il peut y avoir ambiguïté sur ce à quoi doit s'appliquer une valeur.

Dans ce cas-là, il faudra respecter un ordre précis. J'indiquerai l'ordre de déclaration pour chaque notation raccourcie que nous allons étudier dans ce cours dans la leçon qui lui est relative.

Par exemple, la propriété raccourcie padding va permettre de définir le comportement des marges internes d'un élément HTML. Cette propriété est la version short hand des propriétés suivantes :

- padding-top : définition de la marge interne supérieure ;
- padding-right : définition de la marge interne droite ;
- padding-bottom : définition de la marge interne inférieure ;
- padding-left : définition de la marge interne gauche.

Chacune de ces quatre propriétés va accepter le même type de valeur et notamment des valeurs de type « longueur » en px par exemple. Lors de la définition de padding, il faudra donc faire attention à l'ordre des valeurs pour définir la bonne taille pour chacune des marges internes.

## Que se passe-t-il en cas d'oubli de déclaration de certaines valeurs dans les notations raccourcies ?

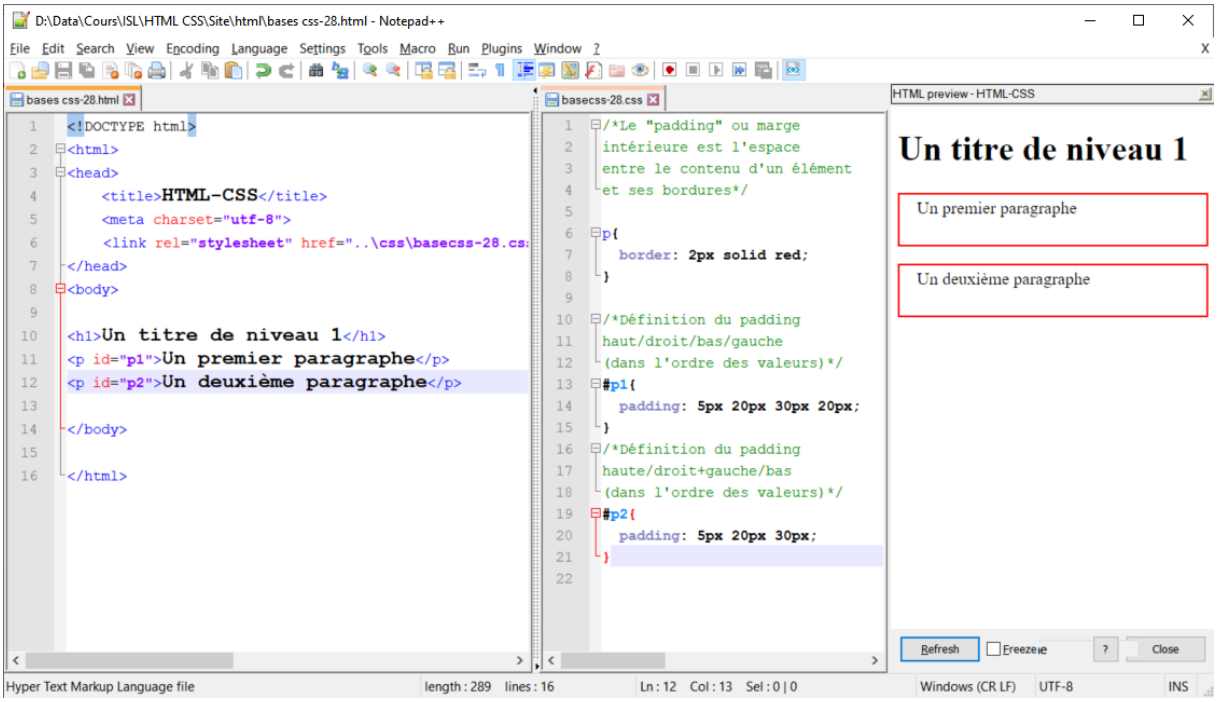
Avant tout, vous devez bien comprendre que nous ne sommes jamais obligés de définir le comportement de chacune des propriétés long hand dans la propriété short hand associée.

En d'autres termes, on va tout à fait pouvoir omettre de déclarer certaines valeurs dans nos propriétés raccourcies.

Ici, il va y avoir principalement deux cas à distinguer : le cas où il peut y avoir ambiguïté sur les valeurs et le cas où il n'y en a pas.

Dans le cas où il peut y avoir ambiguïté, la définition de la propriété raccourcie va nous indiquer son comportement. Prenons l'exemple de notre propriété padding par exemple. Si on omet une valeur, la propriété ne peut pas savoir à première vue si c'est la valeur de la marge haute, droite, basse ou gauche que l'on ne souhaite pas définir.

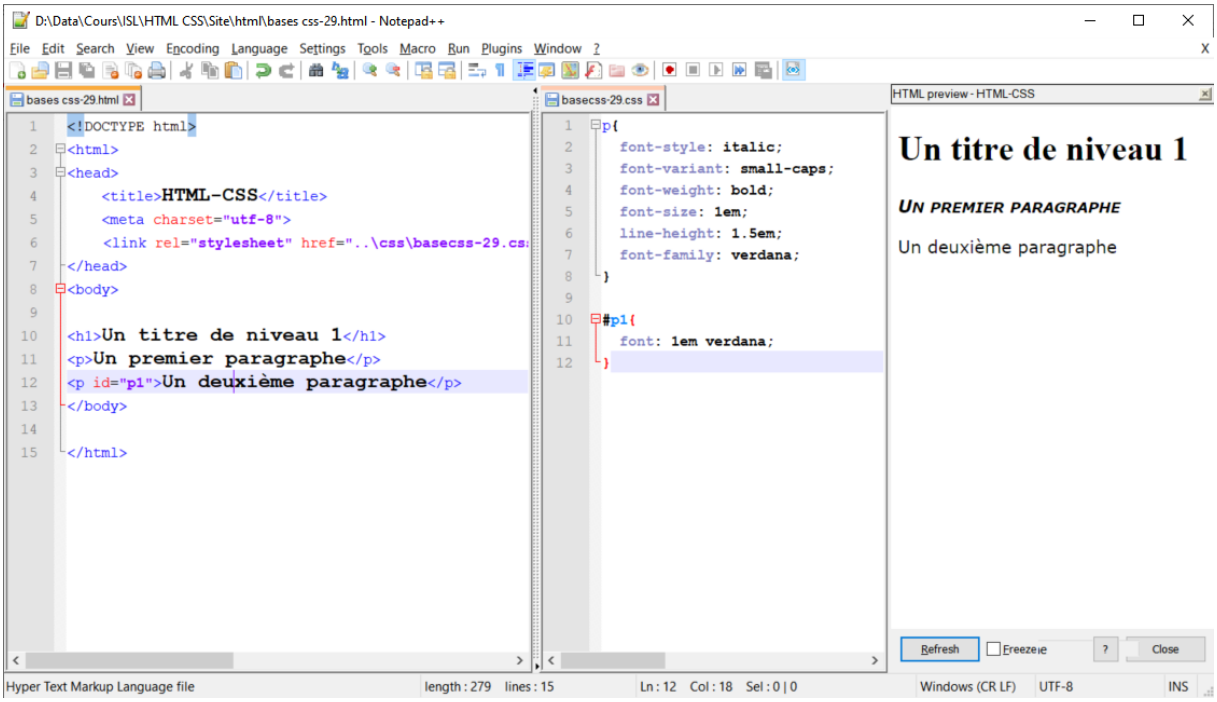
Pour gérer ce genre de situations, des règles ont donc été définies lors de la création de la version raccourcie. Dans le cas de padding, par exemple, si on ne passe que trois valeurs alors la deuxième va s'appliquer à la fois à la marge droite et à la marge gauche.




Dans le cas où il n'y a pas d'ambiguïté, ce sont les valeurs par défaut des propriétés relatives qui vont être utilisées. Cela signifie que les valeurs qui ne sont pas définies avec la propriété raccourcie sont définies avec leur valeur initiale.

Faites bien attention donc ici car omettre des valeurs dans une propriété raccourcie va tout de même définir le comportement des propriétés relatives avec leur valeur initiale.

En particulier, dans le cas où la propriété en question avait déjà été définie auparavant avec sa notation long hand, la valeur sera surchargée par la propriété raccourcie ce qui signifie que c'est la valeur par défaut transmise par la version raccourcie qui sera utilisée. Faites donc bien attention aux comportements inattendus !




Dans l'exemple ci-dessus, par exemple, on définit des comportements pour chacune des propriétés qui peuvent être définies avec la notation raccourcie font pour nos éléments p.

Ensuite, on utilise la notation raccourcie font en ciblant un paragraphe en particulier. Dans font, on ne définit que les valeurs relatives à la taille de la police et à la famille de polices utilisée.

Pour toutes les autres valeurs, ce sont les valeurs par défaut qui vont être utilisées. Ainsi, par exemple, le style (font-style) et le poids (font-weight) vont être normal, ce qui est la valeur par défaut de ces deux propriétés et non pas italic et bold comme on l'a défini au-dessus.

## Les limites des propriétés short hand par rapport aux notations long hand

La première limite des propriétés short hand par rapport à leurs équivalentes long hand est qu'on ne va pas pouvoir utiliser les valeurs globales inherit, initial et unset dans la déclaration des valeurs de nos propriétés raccourcies.

En effet, si on les utilisait, il serait impossible pour les navigateurs de savoir à quelle propriété correspond quelle valeur dans le cas de l'oubli de certaines valeurs.

Une deuxième limite évidente est que l'héritage des propriétés de va pas être possible avec les propriétés raccourcies puisque les valeurs oubliées dans les propriétés raccourcies vont être remplacées par leurs valeurs initiales. Il ne va donc pas être possible de pouvoir faire hériter les valeurs de certaines propriétés en les omettant dans la déclaration short hand puisque le CSS va tout de même automatiquement les définir en utilisant les valeurs initiales des propriétés « non définies ».

## Quelques notations short hand courantes et les propriétés long hand associées

Avant tout, notez que chacune des propriétés long hand ne va pas forcément avoir de propriété short hand associée.

Les propriétés short hand ont été créées pour simplifier et raccourcir l'écriture du CSS tout en gardant une cohérence et une bonne lisibilité du code. C'est la raison pour laquelle les propriétés short hand regroupent toujours des ensembles de propriétés qui agissent sur un même aspect d'un élément HTML.

Vous pourrez trouver dans le tableau ci-dessous les propriétés short hand les plus courantes et qu'il vous faut connaitre avec l'ensemble des propriétés long hand qu'elles permettent de définir.

Nous n'avons pour le moment pas étudié la majorité de ces propriétés. Pas d'inquiétude, nous allons le faire au cours de ce cours.

| **Short Hand** | **Equivalent Long Hand** |
|---|---|
| font | font-style, font-variant, font-weight, font-size, line-height, font-family |
| border | border-width, border-style, border-color |
| margin | margin-top, margin-right, margin-bottom, margin-left |
| padding | padding-top, padding-right, padding-bottom, padding-left |
| background | background-image, background-position, background-size, background-repeat, background-origin, background-clip, background-attachment, background-color |
| transition | transition-property, transition-duration, transition-timing-function, transition-delay |
| animation | animation-name, animation-duration, animation-timing-function, animation-delay, animation-iteration-count, animation-direction, animation-fill-mode, animation-play-state |
| flex | flex-shrink, flex-grow, flex-basis |