



Refcard

Redécouvrir CSS

Cécile HUI-BON-HOA
Mars 2019



INTRODUCTION

Aujourd'hui, même si le web ne se résume pas uniquement au trio HTML-CSS-Javascript, il reste indispensable de maîtriser ces langages. Malgré cela, la plupart des développeurs se désintéressent de CSS au profit d'autres technologies, CSS n'attirant l'attention que lorsqu'il s'agira d'aligner des éléments, de rendre un site responsive ou juste d'essayer de rendre le côté front plus attractif.

À travers cette refcard, j'aimerais vous montrer une autre facette de ce langage, partager avec vous certaines bonnes pratiques, astuces et peut-être vous prouver que ce langage mérite votre attention.

DÉFINITION

CSS, de l'anglais Cascading Style Sheets, est un langage qui décrit la présentation de pages web. Il permet notamment d'adapter la présentation de ces pages à des écrans de tailles différentes.

Rappels

Tout d'abord, rappelons qu'écrire du CSS revient à écrire une liste de règles de style que le navigateur va interpréter.

Une règle CSS ressemble à ceci :

H1 {color: red}

Sélecteur

Déclaration : **color** = propriété
red = valeur

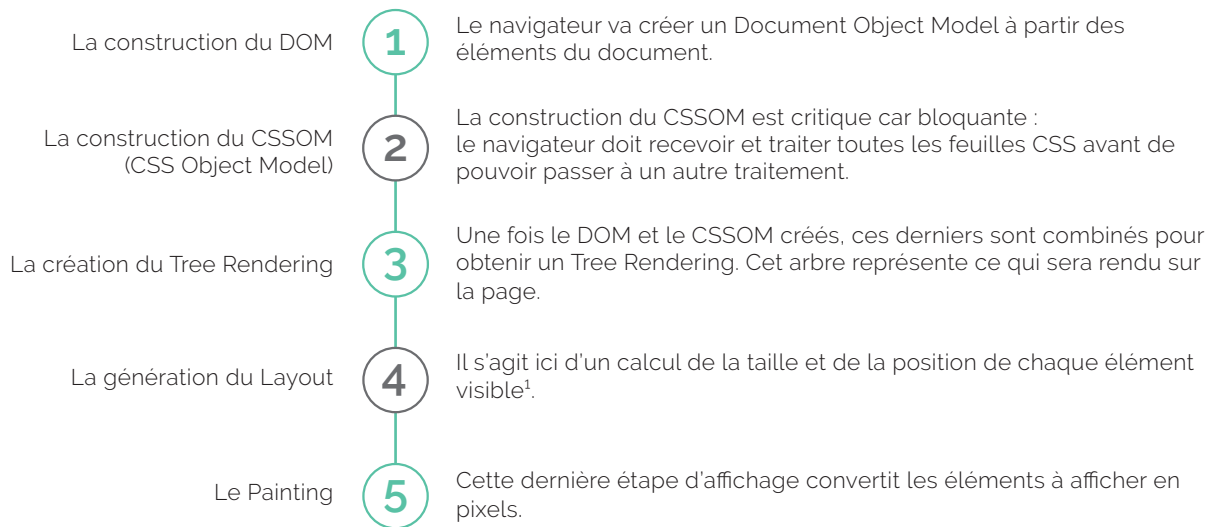
Ce bloc se compose d'un sélecteur (ici **h1**) permettant d'identifier l'élément auquel appliquer le style, et d'une ou plusieurs déclarations (ici **color: red**).

Une déclaration décrit comment une caractéristique visuelle devrait s'afficher. Elle comporte une propriété et une valeur. Ici nous donnons à la propriété **color** la valeur **red**. Les éléments **h1** seront donc affichés en rouge.



L'affichage par les navigateurs

Pour afficher une page, les navigateurs exécutent un traitement appelé chemin critique de rendu (ou critical rendering path). Ce travail peut être découpé en 5 étapes.



Afin de ne pas ralentir cet enchaînement, il est important d'écrire des feuilles de style simples. On parle de feuilles de style en cascade, ce qui implique une hiérarchie et un héritage des propriétés qu'il est nécessaire de garder en tête.


¹ La notion de visible renvoie aux éléments qui occupent un espace dans la mise en page. Ainsi, les éléments avec la propriété **visibility: hidden** seront invisibles mais présents sur la page, tandis que les éléments ayant la propriété **display: none** ne seront pas calculés car ils seront supprimés de l'arborescence.

Bien choisir son sélecteur

Pour qu'un style CSS soit appliqué à un élément du DOM, le navigateur doit pouvoir faire le lien entre les deux (cf. étape de Tree Rendering). Cette association s'appuie sur un sélecteur dont il existe plusieurs types.

NOM	PATTERN	DÉFINITION	EXEMPLE
Universal	*	Correspond à tout élément. <i>À noter que le style sera appliqué à tous les éléments. Il est donc à utiliser avec précaution.</i>	<code>* { color: green; }</code>
Type	E	Correspond à tout élément de ce type	<code>h1 { color: blue; }</code>
Grouping	E, F, G	Lorsque plusieurs éléments partagent les mêmes déclarations, nous pouvons les regrouper en les séparant par des virgules.	<code>h1, h2, h3 { color: blue; }</code>
Class	.maClasseDeStyle	Correspond à tout élément dont la classe est "maClasseDeStyle"	<code>.maClasseDeStyle { color: blue; }</code>
ID	#monIdentifiant	Correspond à tout élément dont l'ID est "monIdentifiant"	<code>#monIdentifiant { color: blue; }</code>
Child	E > F	Correspond à tout élément F enfant direct (dans le DOM) d'un élément E	<code>div > p { color: blue; }</code>



NOM	PATTERN	DÉFINITION	EXEMPLE
Descendant	E F	Correspond à tout élément F enfant (dans le DOM) d'un élément E, quel que soit le niveau de descendance.	<code>h1 span { color: blue; }</code>
Adjacent sibling	E + F	Correspond à tout élément F ayant le même parent que l'élément E, l'élément F devant suivre directement l'élément E.	<code>h1 + p { color: blue; }</code>
General sibling	E ~ F	Correspond à tout élément F ayant le même parent que l'élément E	<code>h1 ~ p { color: blue; }</code>
Attribute	<div>E[att] E[att="val"] E[att~="val"] E[att]="val" E[att^="val"] E[att\$="val"] E[att*="val"]</div>	<p>Il existe plusieurs façons de créer un sélecteur en utilisant les attributs :</p> <p>E[att] : les éléments dont l'attribut "att" est présent</p> <p>E[att="val"] : les éléments dont la valeur de l'attribut "att" est égale à "val"</p> <p>E[att~="val"] : les éléments dont l'attribut "att" est une liste de mots séparés par des espaces contenant "val"</p> <p>E[att]="val" : les éléments dont l'attribut "att" vaut "val" ou commence par "val" suivi d'un tiret ("val-")</p> <p>E[att^="val"] : les éléments dont l'attribut "att" commence par "val"</p> <p>E[att\$="val"] : les éléments dont l'attribut "att" termine par "val"</p> <p>E[att*="val"] : les éléments dont l'attribut "att" contient le texte "val"</p> <p> Astuce mnémotechnique pour les adeptes d'expressions régulières : les caractères ^, \$ et * s'emploient de la même manière.</p>	<pre>a[disabled] {...} a[target="_blank"] {...} a[class~="btn"] {...} // Ex: <a lang="en-S"... a[lang]="en" {...} a[href^="https://"] {...} a[href\$=".com"] {...} a[class*=text] {...}</pre>

Le cas des pseudo-éléments et des pseudo-classes

Il est également possible d'utiliser des pseudo-éléments et des pseudo-classes comme sélecteurs.

- Un pseudo-élément permet d'affiner la sélection sur l'élément ciblé par la règle. Par exemple, le bloc `h1:first-letter {color: red;}` affichera la première lettre des éléments h1 en rouge.

Titre

- Une pseudo-classe indique l'état dans lequel l'élément doit être pour que le style soit appliqué. Il existe l'état statique (comme `:visited`) ou dynamique (comme `:hover`). Ainsi, le bloc `a:visited{color:green;}` affichera les liens visités en vert.

lien vers un site visité



La spécificité hiérarchique

À chaque sélecteur correspond une valeur qui est additive : une combinaison de sélecteurs vaut la somme de leur valeur respective. Les navigateurs utilisent ces valeurs comme poids pour établir une hiérarchie d'application des sélecteurs.

 Avoir ce mécanisme en tête permet de comprendre pourquoi certains styles sont appliqués et d'autres non.

Les sélecteurs peuvent être classifiés en 4 groupes, du plus au moins important :

► L'attribut style (A)

```
<h1 style="color: green;">Mon Titre</h1>
```

► Les identifiants (B)

```
#monIdentifiant { color: blue; }
```

► Les classes, pseudo-classes et autres attributs (C)

```
.maClasseDeStyle { color: blue; }
```

► Les éléments et pseudo-éléments (D)

```
h1 { color: green; }
```

À noter que le sélecteur universel est ignoré et donne une valeur de 0. La répétition du même sélecteur est autorisée et augmente la spécificité. Par ailleurs, lorsque plusieurs sélecteurs ayant une spécificité égale sont en concurrence sur une propriété, l'ordre des déclarations sera pris en compte et c'est la dernière valeur qui sera appliquée.

Pour calculer la valeur d'un sélecteur, il faut comptabiliser les sélecteurs de chaque groupe et les concaténer.

EXEMPLE	EXPLICATION	SPÉCIFICITÉ
*	Sélecteur universel : A=0 B=0 C=0 D=0	0
LI	1 élément : A=0 B=0 C=0 D=1	1
UL LI	2 éléments : A=0 B=0 C=0 D=2	2
UL OL+LI	3 éléments : A=0 B=0 C=0 D=3	3
H1 + *[REL=up]	1 élément et 1 attribut : A=0 B=0 C=1 D=1	11
UL OL LI.red	3 éléments et 1 classe : A=0 B=0 C=1 D=3	13
LI.red.level	1 élément et 2 classes : A=0 B=0 C=2 D=1	21
#x34y	1 identifiant : A=0 B=1 C=0 D=0	100
#s12:not(F00)	1 identifiant et 1 attribut : A=0 B=1 C=0 D=1	101



Les bonnes pratiques

ÉVITER D'UTILISER !IMPORTANT

Lors du calcul de la spécificité hiérarchique, le sélecteur universel est ignoré. En revanche, n'importe quelle déclaration utilisant le mot clé **!important** aura la spécificité la plus haute et sera prioritaire par rapport à n'importe quelle autre règle sur cette propriété. Le seul moyen de contrer cela est de définir un sélecteur avec une plus grande spécificité et d'y déclarer le mot clé **!important**.

Par exemple la règle : `.greenTitle{ color: green !important;}` a une spécificité de 10 et peut être outrepassée par une spécificité > 10 comme par exemple : `H1.bluetitle{ color: blue !important;}` qui a une spécificité de 11.

Même si cela ne semble pas être un problème très compliqué à résoudre, il faut avoir conscience que cette solution sera à mettre en place à chaque fois que ce mot clé impactera une règle que vous souhaitez modifier. Au fil du temps, vous créerez un ensemble de règles comportant de plus en plus de mots clés **!important** et il vous sera difficile de faire évoluer les styles appliqués.

Pour éviter de se retrouver pris dans cette spirale, le mieux est de choisir sciemment les sélecteurs en maîtrisant la spécificité hiérarchique.

LIMITER LE NOMBRE DE SÉLECTEURS

Pour des raisons de performances, les navigateurs lisent les sélecteurs de droite à gauche. La majorité des sélecteurs ne correspondront pas à l'élément et ce sens de lecture permet de tomber en échec plus rapidement. C'est pourquoi il est important de garder un CSS concis, générique et réutilisable.

Créer des sélecteurs trop précis, en renseignant trop d'éléments par exemple, va retarder cette mise en échec et rendre l'affichage plus lent. Pour le sélecteur **H1.bluetitle**, le navigateur va chercher les éléments avec la classe **bluetitle**. Puis il cherchera parmi ces éléments ceux qui sont de type **H1**. À chaque fois qu'un sélecteur est ajouté, on augmente inutilement la phase de rendu. En utilisant correctement les sélecteurs, on évite cet éventuel ralentissement et améliore la lisibilité du CSS.

LA RÈGLE INCEPTION

Visant spécifiquement l'utilisation abusive d'imbrication de sélecteurs, la règle **Inception** (ou **Inception Rule**) est de plus en plus répandue. Cette règle vise à éviter le plus possible d'avoir plus de 3 niveaux hiérarchiques de sélecteurs. Inscrire une forte hiérarchie dans les sélecteurs réduit la rapidité du rendu, rallonge la taille des fichiers et le temps de maintenance.

```
body { ... }
body div.content div.container { ... }
body div.content div.container div.items { ... }
body div.content div.container div.items > div.description { ... }
body div.content div.container div.items > div.description div.title { ... }
body div.content div.container div.items > div.description div.title h1 { ... }
```

...

Un contenu difficile à lire et à maintenir

En les définissant ainsi, les styles sont liés exclusivement à cette structure HTML, ce qui va complètement à l'encontre de l'aspect «Cascade» du CSS.



Garder un CSS concis, lisible, générique et réutilisable

UTILISER PLUSIEURS FEUILLES CSS

Il est souvent préférable de créer une feuille de style globale puis plusieurs feuilles de style, chacune ayant une thématique spécifique. Des feuilles de styles plus courtes et plus ciblées seront plus faciles à documenter (ou implicitement documentées par la thématique de leur contenu). Cette pratique permet de s'y retrouver plus facilement et ainsi réduire le temps de recherche d'un style spécifique.

Avec l'arrivée d'HTTP2 et de son multiplexing, cette pratique a encore plus de sens car seul le CSS nécessaire sera chargé sur la page. Mais il faut noter que pour les navigateurs ne supportant pas encore HTTP2, chaque feuille CSS liée à une page HTML aura pour conséquence une requête HTTP pour la récupérer. Afin de limiter le nombre d'appels, il peut être judicieux d'utiliser un outil qui combinera toutes ces feuilles de style en une seule. La minification de la feuille de style résultante pourra également optimiser le temps de chargement.

AMÉLIORER LA LISIBILITÉ EN NOMMANT CORRECTEMENT LES CLASSES

Le nom d'une classe devrait permettre aux autres acteurs du projet de comprendre exactement ce qu'elle fait et quand l'utiliser. Sans expliciter tous les détails, ces noms doivent rester descriptifs. Il faut donc trouver un juste milieu afin d'obtenir des noms précis et explicites tout en restant concis.

Les abréviations sont à éviter sauf exception. Ainsi, préférez "checkbox" à "chk". En revanche, "btn" est acceptable tant il est courant pour le rendu de boutons. Dans tous les cas, le plus important est de rester homogène dans la façon de nommer les classes. Si vous avez déjà des classes `btn-primary` et `btn-secondary`, restez dans cette nomenclature et évitez de créer une classe `cancel-btn`.

Pour une lecture plus facile, l'usage est d'utiliser des tirets (-) dans les noms. Les underscores (_) n'ayant pas été supportés dès le début, le tiret est devenu une norme. Mais il existe certaines méthodes de nommage qui utilisent l'underscore, comme BEM. La meilleure méthode étant de trouver celle qui vous convient et facilite le travail collaboratif.

COMMENTER LE CODE

Comme avec n'importe quel langage de programmation, les commentaires peuvent apporter une facilité de lecture et de compréhension. Ils permettent de fournir le contexte d'utilisation. Ils peuvent ainsi expliquer un style en particulier peu explicite, fournir un exemple de code HTML ou un lien vers une page utilisant la feuille de style.

Les commentaires peuvent aussi être descriptifs. Cependant, on utilisera en premier lieu les autres bonnes pratiques, dans la mesure où elles sont applicables au projet :

- Le nommage du fichier permet de décrire son contenu. Combiné à un gestionnaire de code source, cela permet d'éviter les pavés de commentaires en entête avec auteurs, description et historique des modifications.
- Le découpage en plusieurs petits fichiers et l'arborescence résultante donnent un résultat plus pérenne que l'emploi de commentaires pour délimiter des sections.

Les commentaires présentent des inconvénients à long terme : ils représentent un surcoût de maintenance et un surcroît de discipline pour les tenir à jour et les garder liés au code commenté. Mais, tôt ou tard, ils ne le seront plus et alors ils deviendront au mieux obsolètes, au pire pourront induire en erreur.



CLEAN

Ordonner les propriétés par ordre alphabétique est une pratique qui vous permettra de trouver plus rapidement une propriété ou de détecter une erreur. Valider son CSS est une nécessité absolue, cela permettra de vérifier qu'il n'y a pas d'erreur et que les spécificités du langage sont respectées. La plupart des IDE permettent de valider une feuille de style, mais vous pouvez également utiliser un validateur en ligne.

Vous pouvez aller encore plus loin et utiliser un linter qui vérifiera la qualité du code et pourra proposer des optimisations, comme celle de ne pas associer d'unité aux valeurs 0. Pour ne pas subir cet outil, tournez-vous plutôt vers une solution qui permet de customiser entièrement les règles appliquées comme CSSLint.

L'utilisation de préprocesseurs CSS comme Sass, Less ou Stylus est de plus en plus courante. Ces outils ajoutent des fonctionnalités qui n'existent pas en CSS pur, comme l'imbrication ou l'héritage de sélecteurs. L'objectif est de rendre le code plus lisible et plus facile à maintenir. Mais cette pratique n'est pas une nécessité car il vaut mieux maîtriser CSS avant de se lancer dans ce nouveau procédé.

REDUCE, REUSE, RECYCLE

Certains éléments peuvent parfois partager les mêmes propriétés. Plutôt que de répéter les mêmes blocs pour des éléments différents, les regrouper rend le code plus lisible.

Ainsi,

```
h1 {  
  font-family: tahoma;  
  color: grey;  
}  
h2 {  
  font-family: tahoma;  
  color: grey;  
}
```

devient

```
h1, h2 {  
  font-family: tahoma;  
  color: grey;  
}
```

De même, CSS permet de combiner certains styles pour certaines propriétés comme **padding**, **margin**, **border** etc. Par exemple, la propriété **border** permet de regrouper les propriétés **border-style**, **border-color** et **border-width**.

Ainsi,

```
div {  
  border-style: solid;  
  border-color: #FF4D80;  
  border-width: 1px;  
}
```

devient

```
div {  
  border: solid 1px #FF4D80;  
}
```

Les bonnes pratiques de code s'appliquent également au CSS. Il est d'usage de refactorer son code pour remplacer les duplications par un bloc réutilisable. Par ailleurs, l'attribut style ne permettant pas une réutilisation de son contenu, on limitera son utilisation au maximum. On note également une volonté de rendre ce langage modulaire avec des méthodes comme SMACSS, OOCSS, ITCSS ou Atomic design.



Astuces

LES POLICES D'ÉCRITURE

```
H1 {font-family: Roboto, Arial, Sans-serif;}
```

Il est préférable de proposer plusieurs polices car la propriété **font-family** fonctionne sur un système de fallback : si la première police n'est pas supportée par le navigateur, il essaiera d'utiliser la deuxième et ainsi de suite. Terminer par un type de police permet au navigateur de choisir une police de ce type parmi celles dont il dispose.

En effet, il existe 3 principaux types de police différents : serif, sans-serif et monospace. Les polices de type serif sont celles avec empattement. Les polices de type sans-serif sont celles qui ne possèdent pas d'empattement. Tandis que les polices de type monospace ont la particularité d'avoir chacun de leur caractère de la même largeur.

SERIF	SANS-SERIF	MONOSPACE
Times new Roman	Arial	Courier New

LES UNITÉS

Certaines propriétés permettent d'utiliser différents types d'unités : relatives comme pixel (px) et absolues comme pourcentage (%), viewport width (vw), etc. Les unités **em** et **rem** (pour root em) sont celles recommandées par le W3C. 1rem équivaut à la taille de l'élément racine de la page, tandis qu'1 em peut être différent pour chaque élément : si un élément a une taille de 16px alors 1em équivaut à 16px pour cet élément.

Ces unités relatives ont l'avantage de donner aux développeurs une plus grande flexibilité pour ajuster le design et le rendre plus facilement responsive. Par ailleurs, les utilisateurs ayant spécifié une **font-size** différente (plus grande pour un meilleur confort visuel par exemple) auront un contenu affiché selon leurs préférences.

DIFFÉRENCIER LE BLOC ENGLOBANT DE SON CONTENU

Pour garder des styles facilement réutilisables, il est important de différencier le style d'un élément et celui de son bloc englobant (ou conteneur). L'élément portera les propriétés propres à sa fonction tandis que son conteneur portera les propriétés propres au contexte. Ainsi un bouton d'action pourra avoir la propriété **color: blue** et son conteneur la propriété **margin: 1em**.

Autre exemple classique : pour qu'un élément soit en **position: absolute**, il faut que son conteneur soit en **position: relative**.



BOX-SIZING

Utiliser des boxes n'est pas toujours facile. Lorsque l'on veut aligner plusieurs boxes sur la même ligne, on leur attribue par exemple la largeur de 25% pour en positionner 4. Mais il faut savoir que la largeur d'une box est calculée de la manière suivante :

largeur + bordure gauche + bordure droite + padding gauche + padding droit.

Si nous voulons ajouter une bordure ou un padding à nos boxes, elles ne tiendront plus sur la même ligne.



Ces 4 boxes de largeur 25% ayant des bordures de tailles différentes n'ont pas une largeur finale de 25% et ne tiennent pas sur la même ligne

Une méthode permettant de faciliter le layout des boxes est d'appliquer la valeur `border-box` à la propriété **box-sizing**. Toutes les propriétés CSS n'appliquent pas le principe de cascade. La propriété **box-sizing** en fait partie, sa valeur par défaut est **content-box**. C'est pourquoi il est nécessaire d'appliquer cette valeur à tous les éléments.

```
/* apply a natural box layout model to all elements, but allowing components to change */
html {
  box-sizing: border-box;
}
*, *:before, *:after {
  box-sizing: inherit;
}
```

*Recommandation de Paul Irish pour appliquer la propriété **box-sizing** : **border-box** à tous les éléments*



En fixant la largeur à 25%, peu importe la taille de la bordure, nos 4 boxes font la même largeur



Paul Irish est développeur front-end et travaille actuellement dans l'équipe Google Chrome <https://www.paulirish.com/>.



LA NORMALISATION

La plupart des navigateurs ont tendance à afficher certains styles CSS de différentes manières. Ainsi, un même site peut avoir une apparence légèrement différente dans Chrome, Safari ou Firefox. Pour éviter cette hétérogénéité, il est possible d'utiliser une feuille de style qui corrigera ces disparités. Plusieurs solutions existent comme Normalize¹ ou Reset² mais attention car ces outils corrigeront toutes les dissemblances.

Un grand nombre d'éléments seront impactés et toutes ces corrections ne seront pas utiles à votre projet. Il reste préférable de s'appuyer sur ces outils pour ne prendre que les régularisations nécessaires. Cela vous permettra encore une fois de garder un CSS concis et lisible.

¹ Le projet Normalize est disponible sur github : <https://necolas.github.io/normalize.css/> Il permet aux navigateurs de rendre tous les éléments de manière plus consistante et suivant les normes actuelles. Il cible les styles qui ont besoin d'être normalisés.

² Le projet Reset est disponible ici : <https://meyerweb.com/eric/tools/css/reset/> Ce projet se veut volontairement très générique et est une véritable réinitialisation, comme si vous ne partiez de rien.

La compatibilité entre différents navigateurs

LES PRÉFIXES VENDEURS (*VENDOR PREFIXES*)

Pour assurer la compatibilité au plus grand nombre d'utilisateurs, il existe des propriétés spécifiques pour celles qui ne sont pas supportées nativement par certains navigateurs. Il s'agit de "propriétés propriétaires" ou "préfixes vendeurs" qui sont des sortes de mots-clés devant être préfixées d'un tiret et d'un code correspondant au moteur de rendu les exploitant.

```
selecteur {  
  -webkit-property : valeur; /* pour Chrome, Safari, Android */  
  -moz-property : valeur; /* pour Firefox */  
  -ms-property : valeur; /* pour Internet Explorer */  
  -o-property : valeur; /* pour Opera */  
  property : valeur;  
}
```

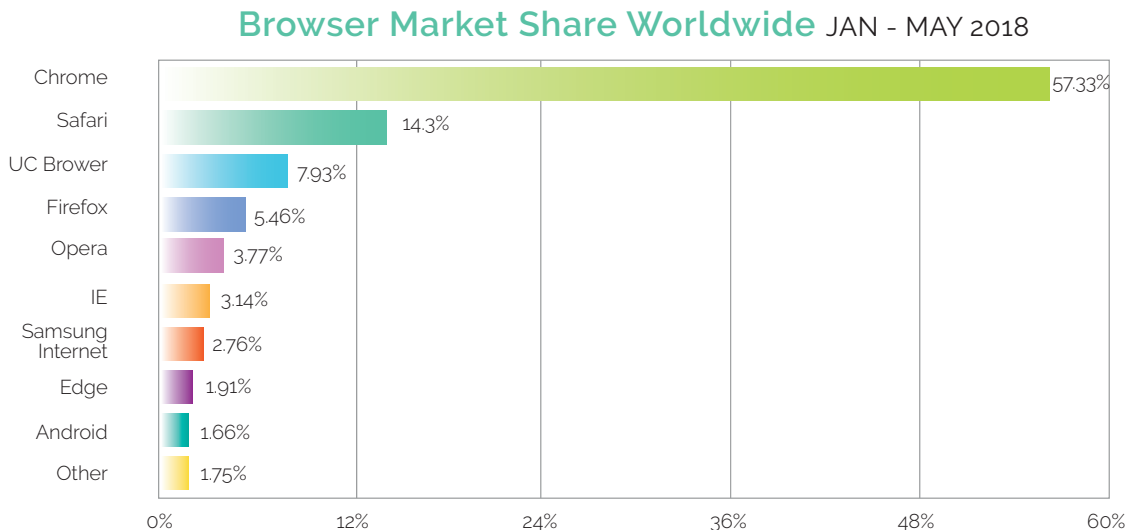
Pour assurer le rendu de la propriété `border radius` par tous les navigateurs, le style pourrait donc être décrit comme ceci :

```
.cercle {  
  -webkit-border-radius : 50%;  
  -moz-border-radius : 50%;  
  border-radius : 50%;  
}
```

À noter qu'il existe des plugins comme `autoprefixer` qui permettent de parser votre CSS et d'y ajouter les préfixes vendeurs nécessaires.



Il n'est pas nécessaire de supporter tous les navigateurs. Il existe des statistiques sur lesquelles vous pouvez vous baser pour choisir les navigateurs à prendre en compte. Ces statistiques peuvent être affinées par système d'exploitation, région du monde, etc.



Statistiques d'utilisation mondiale de navigateurs de janvier 2018 à mai 2018 par <http://gs.statcounter.com/>

Certaines propriétés ne sont pas disponibles pour tous les navigateurs. Le site <https://caniuse.com/> répertorie un grand nombre de ces propriétés et leur support ou non par les navigateurs et selon leur version.

NB : pour chaque version de navigateur apparaît le pourcentage d'utilisation, ce qui permet de savoir plus facilement s'il est intéressant d'utiliser cette propriété.

Ordonner les propriétés

Il est à souligner qu'une vérification régulière de ces préfixes est à faire. Une fois que les navigateurs intègrent les nouvelles fonctionnalités CSS, ces propriétés spécifiques deviennent inutiles.

Actuellement, elles ne sont pas pour autant ignorées. Lorsque le navigateur prend en charge la propriété mais que son équivalent est renseigné, c'est la dernière propriété déclarée qui sera appliquée.

```
.vendor-prefix-after {  
  border-radius: 10px;  
  -webkit-border-radius: 50px;  
}
```

vendor-prefix-after

```
.vendor-prefix-before {  
  -webkit-border-radius: 50px ;  
  border-radius: 10px;  
}
```

vendor-prefix-before



Lorsque le préfixe vendeur et la propriété native sont renseignés, c'est la dernière propriété déclarée qui prend le dessus.



En théorie, les deux propriétés sont censées avoir un rendu identique. En réalité, certains préfixes vendeurs n'ont pas le même comportement que les propriétés natives. Cela peut entraîner une disparité selon le navigateur utilisé.

```
vendor-prefix-after {  
  border-radius: 50px 10px;  
  -webkit-border-radius: 50px 10px;  
}
```

vendor-prefix-after

```
.vendor-prefix-before {  
  -webkit-border-radius: 50px 10px;  
  border-radius: 50px 10px;  
}
```

vendor-prefix-before

Les deux propriétés n'interprètent pas les valeurs de la même manière et brisent la continuité du rendu d'un navigateur à l'autre.

Les animations CSS

Pour améliorer l'expérience utilisateur, nous pouvons ajouter des animations CSS. Elles pourront alors informer l'utilisateur (lors d'un chargement d'information par exemple) ou le guider lors d'un processus (avec un guide de prise en main lors de la première utilisation par exemple). Il faut distinguer les transitions qui permettent de créer une transition d'un état à un autre et les animations qui permettent de créer une série de mouvements plus complexes.

Toutes les propriétés ne peuvent être animées¹. Il est préférable d'utiliser les propriétés **transform** et **opacity** plutôt que de modifier directement les propriétés **width**, **height**, **left**, **top**, **bottom**, **right**. Cela permettra d'avoir des animations plus fluides et une feuille de style plus facile à lire. Grâce aux propriétés **transform** et **opacity**, il est facile de jouer sur la position, l'échelle, la rotation et l'opacité d'un élément. Par exemple, le déplacement d'un élément sera plus facile à interpréter pour les navigateurs avec les propriétés **transform: translateX** et **transform: translateY** plutôt que qu'avec les marges ou paddings. Cela s'explique par le fonctionnement du chemin critique de rendu expliqué au début. L'utilisation de certaines propriétés comme **transform** et **opacity** ne nécessite pas la reconstruction entière de la page contrairement à d'autres.

Pour ne pas disperser l'attention de l'utilisateur, il est préférable de limiter le nombre d'animations sur la page. L'idéal étant de chorégraphier ses animations, en ne les démarrant pas toutes au même moment, ne pas leur donner la même durée etc. Cela permettra d'avoir une meilleure cohérence et un plus grand impact.

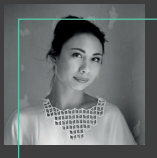
Il est également à noter que pour obtenir une animation fluide, il faudrait avoir 60fps (frame per second), ce qui veut dire que chaque frame doit être rendu en moins de 16ms. Il est donc d'autant plus important de prêter attention à la performance du rendu de l'animation.

¹Certains sites listent les propriétés pouvant être animées comme https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties, et le projet <http://leaverou.github.io/animatable/> propose une visualisation d'animations de certaines de ces propriétés.



CONCLUSION

À travers cette refcard, vous avez pu découvrir qu'à l'instar de n'importe quel langage, l'utilisation de CSS peut être facilitée par l'application de bonnes pratiques. Cela vous prendra peut être du temps avant que tout cela ne devienne des réflexes mais les mettre en pratique vous facilitera le quotidien et améliorera la qualité de vos projets.



Cécile HUI-BON-HOA

Ingénieure Full Stack JS - SOAT

☞☞ Débutant ma carrière en tant qu'ingénieure back-end, j'ai ensuite évolué vers un profil Fullstack me permettant de comprendre et maîtriser la donnée à chaque étape. Manipuler au quotidien une grande volumétrie de données m'a encouragé à m'appuyer sur la data visualisation pour créer des vues de données adaptées. Afin de répondre à ces nouveaux besoins en créant des outils personnalisés, la maîtrise du trio HTML-CSS-Javascript reste indispensable. ☞☞

SOAT

89 quai Panhard et Levassor – 75013 Paris

Tél. : + (33) 1.44.75.42.55

contact@soat.fr - www.soat.fr

Retrouvez-nous sur

