#10 Script.css : CSS

Bonjour! Bon, ça fait quoi, 3 mois que j'ai pas sortie d'épisodes! En effet! Et j'en suis désolé, mais aujourd'hui je retourne au turbin pour vous parler du CSS. En effet, toujours dans cette idée de parler des principaux langages du web, il est l'un des plus important puisque c'est lui qui va rendre votre page web belle et agréable. Et on rentrera dans la partie programmation du domaine une prochaine fois, quand il sera temps de parler du JS!

Mais du coup on va commencer par voir un peu qu'est ce que le CSS, quelle est son histoire. Ensuite on parlera un peu de comment il fonctionne techniquement, et on terminera par voir quelles sont les améliorations qui arrivent par rapport au CSS.

Commençons! Le nom complet du CSS est Cascade Style Sheet, soit Feuille de Style en Cascade. Ce nom n'a rien d'anodin puisque le CSS est le langage utilisé pour styliser vos fichiers HTML et définir comment ils doivent s'afficher. Et obéissent à des règles définies en cascade. C'est à dire que ces règles de styles pourront être définie à plusieurs endroits et se chevaucher entre elles, mais s'appliqueront en cascade : celles qui sont définies en premier peuvent êtres annulées par les dernières.

L'idée d'avoir des outils de mise en page à germer dès le début du développement du World Wide Web, mais plus sur le côté utilisateur, où chacun peut décider du style à mettre en place. Mais c'est par la suite que le principe de feuille de style du côté du site arriva. Il y eu différentes propositions de la part des différents navigateurs ou acteurs du domaine. Par exemple Pei Wei proposa le "Stylesheet proposal", Robert Raisch quand à lui fit les "Stylesheets for HTML" et enfin les "Cascading HTML Style Sheet" de Hakon Wium Le. Il y a aussi une tout autre alternative proposée, Netscape travaillant sur les "JavaScript Based Style Sheet", qui devaient permettre de à la fois transformer les documents et de les mettre en page.

Il y avait donc trois problématiques importantes à étudier. Tout d'abord est-ce que le HTML doit permettre juste la structuration des documents, ou alors la structuration et la mise en page? Est ce que la mise en page voulue par l'auteur du site doit avoir une plus grande importance que celle choisie par l'utilisateur? Et enfin est-ce que la mise en page devrait elle être faite par un langage de description, ou alors de transformation?

Au final c'est la version de Hakon Wium Le qui est selectionnée, en partant du principe que le HTML doit s'occuper uniquement de la structuration, que l'utilisateur ai le choix final du style, et que ce soit un langage de description de mise en page et non de transformation.

Mais ce qui fait que le CSS est spécial repose sur deux choses. Déjà c'est le premier langage à introduire le principe de cascades. Grâce à lui on a un système de hiérarchie dans les styles appliqués, et surtout les styles peuvent venir de différents fichiers. Par exemple vous pourrez avoir une feuille pour l'intégralité de votre site web, puis ensuite une par page. Ou autre manière de penser, vous pourrez avoir une feuille faite par vous, le créateur du site, et une faite par l'utilisateur, avec par exemple, des titres plus gros. C'est là sa grande force.

L'autre spécificité du CSS est son principe de développement. En effet il n'a pas été développé à partir de versions successives, comme la plupart des langages ou logiciels actuellement, mais par niveaux. Pour une comparaison, les versions successives c'est comme un train. Vous démarrez à la première gare, puis vous avancez gare par gare, au fil des version. Cela facilite la prise en charge par les navigateurs puisqu'ils peuvent dire qu'ils supportent une version précise du langage. Mais le CSS fonctionne par niveaux, comme un oignon. Le CSS 1 est le centre de l'oignon, le CSS2, une couche supérieure et le CSS3 une couche encore plus grande. Donc le CSS 3 inclut toutes les possibilité du CSS 2 qui contient aussi toutes les possibilité du CSS 1. Cela permet entre autre un développement parallèle des version, mais ça implique aussi un sacré bordel!

Le CSS1 était la pour deux choses. A la fois pour décrire son propre fonctionnement (comment doivent être exprimés les styles, ou comment faire pour que certaines instructions misent en place par la suite puissent être ignorées par les premières versions, pour gérer une compatibilité descendante). Et à la fois là pour définir quelques balises de base, et surtout de la mise en page de texte (taille, alignement, police…)

Le CSS2 quand à lui agrandit les possibilitées avec par exemple des choix de style en fonction du support, le téléchargement et l'utilisations de certaines polices, ou encore de nouvelles possibilités de prendre en compte les choix de l'utilisateur.

Et enfin le CSS3, la version actuelle du CSS, est devenue complètement modulaire, pour faciliter la mise en place des styles par les navigateurs. Il ajoute par exemple les transitions, les media query (qui permettent de faire du style en fonction de la taille de l'écran par exemple), les fonds, les bordures ou encore les variables.

Mais c'est là qu'on voit aussi un des problèmes qui se passe quand on développe un style en CSS. Il se peut que certaines balises qu'on veut utiliser ou qu'on pense compatible avec tous les navigateurs, ne l'est pas. C'est pour ça qu'il faut faire très attention à ce qu'on peut utiliser ou non, grâce à des sites comme caniuse.com, que je vous met dans la description de l'épisode. Il permet de vous dire à partir de quelle version une certaine balise HTML ou propriété CSS est disponible sur tel navigateur, et à vous de faire le choix de l'utiliser ou non.

Je vais aussi en profiter pour vous parler des préfix d'attributs. Parfois certains attributs ne sont pas encore des standards définies officiellement par le Word Wild Web Consortium (l'entité gérant les langages du web comme l'HTML ou le CSS), mais puisque elles sont là depuis suffisamment longtemps, ou définies de manières suffisamment claires, certains

navigateurs vont déjà les mettre en place. Mais il ne mettront pas directement l'attribut en question, puisqu'il n'est pas encore définit par le W3C. Il va mettre l'attribut préfixé en fonction du navigateur. Par exemple pour Chrome c'est -webkit-, pour Firefox c'est -moz-, ou encore pour Opera c'est -o-. La bonne méthode à faire est de mettre à la fois l'attribut préfixé pour chaque navigateur la supportant, et ensuite la propriété sans préfix. Comme ça, si le navigateur la supporte, vous n'avez pas besoin de modifier votre code dans l'absolue.

Bon maintenant que j'ai bien parlé de son histoire et de ses spécifications, il est temps de passer à un peu plus son fonctionnement.

Sachez que vous pouvez écrire votre CSS de 3 manières différentes. Tout d'abord dans un fichier .css classique, que vous importerez grâce à une balise link dans la partie head de votre document HTML. Si vous ne voyez pas ce que c'est la partie head, je vous invite à écouter l'épisode 8 de ce podcast où j'explique tout ce qu'il y a à savoir sur l'HTML. Après, vous pouvez l'écrire directement dans votre fichier HTML, dans une balise style, et enfin, directement dans les éléments que vous voulez styliser, grâce à l'attribut style, et en les séparant par un point-virgule.

La première méthode étant privilégiée pour amener plus de clarté dans votre page principale.

Et comment l'écrire ce fichus CSS? Et bien le CSS va se composer de différents blocs appelés règles. Chaque bloc à la même tête, avec un sélecteur et des attributs.

Commençons par les sélecteurs! Ils vont définir à quel style s'appliquera ce style en particulier. Ca peut être un sélecteur qui s'appliquera à toutes les balises d'un type, auquel cas vous mettrez juste le nom de la balise. Vous pouvez aussi cibler une class ou un id en particulier, grâce au nom de celle ci précédée par un point pour une classe, et précédée par un # pour un id.

Il existe aussi des sélecteurs un peu particulier puisqu'ils vont permettre de soit sélectionner dynamiquement certains éléments ou alors certaines parties des éléments. On appelle ça des pseudos éléments ou des pseudos classes.

Les pseudos-éléments vont s'appliquer à certaines parties d'un élément. On les écrira collé à l'élément qu'ils complètent, avec deux doubles points devants. Il y a par exemple le first-line qui s'appliquera uniquement à la première ligne de l'élément, ou ::first-letter. Il y en a aussi deux particulier, ::before et ::after, qui prennent chacun un attribut content. Ce content sera affiché avant ou après l'élément sélectionné!

Quant aux pseudos-classes, elles s'appliqueront sur l'élément en entier et servent à préciser dans quel état il doit être pour que le style s'applique. Eux vont s'écrire avec simplement un double point devant. Il y a par exemple le :hover qui s'appliquera uniquement quand l'élément est survolé par la souris, le :first-child qui s'appliquera uniquement si l'élément est premier fils dans la liste des balises, ou enfin :focus qui s'appliquera aux éléments qui sont selectionnés. Mais il y a aussi des pseudos classes spéciales qui prennent des paramètres

entre parenthèses! Par exemple le :not() qui s'appliquera seulement si le sélecteur entre parenthèse n'est pas vrai, ou le :nth-child(x) qui s'appliquera seulement si l'élément est le x-ième enfant.

Je vous met de toute façon dans la description la documentation Mozilla exhaustive de tous les pseudos éléments et pseudo classes.

Mais maintenant qu'on a tous ces sélecteur, il serait temps de les combiner entre eux! Et c'est ce qu'on va faire avec les combinateurs! Je vais pas tous vous les citer, ça ne sert à rien, mais dans la description il y aura une documentation spécifique pour ça. Bon partons du principe que vous avez deux selecteurs, A et B. Si vous mettez A espace B, vous sélectionnerez tous les éléments B qui sont enfant à n'importe quel niveau de A. A > B lui va s'appliquer uniquement pour les B directement enfant de A. Pour le reste, je vous laisserais regarder directement la doc.

Bon maintenant qu'on sélectionne bien il est temps de styliser! Je vais pas faire une liste exhaustive de tous les attributs, il y en a trop, et je les connais pas tous honnêtement. Je vais surtout vous parler de ceux liés aux textes, et des différents types d'affichages, parce que c'est ceux qui vont seront le plus utile au début je pense.

Pour mettre en place un attribut, vous allez ouvrir des accolades après le sélecteur et mettre vos attributs. A chaque fois ça sera un couple clé, deux points, valeurs (parfois plusieurs), et se terminant par un point-virgule. Sachez qu'il existe par exemple l'attribut color qui permet de changer la couleur du texte. Il prend soit un nom de couleur en anglais (red, bleu, gree…), un code hexadécimal précédé d'un # ou encore une "fonction" rgb en lui passant les couleur en nombre. On peut aussi utiliser font-size pour changer la taille (en pixel par exemple), ou text-align pour choisir l'alignement du text (center, left, right, justify…).

On peut aussi changer le type d'affichage d'un élément. Il en existe plusieurs et chaque type de balise à un type d'affichage associé. Il en existe beaucoup mais je vais vous parler de ceux que j'ai le plus utilisé et que je trouve personnellement le plus utile. Il y a déjà le type block. Les blocks apparaîtront les uns en dessous des autres, à la suite.par exemple les balises p ou img sont des balises qui apparaîtront comme ça. Ensuite le type inline, qui fait que ça va s'ajouter directement à la ligne, à la suite du block précédent si c'est aussi un inline. Par exemple les balises a présentent dans un paragraphe sont de type inline. Après ça si vous voulez cacher un élément, vous utiliserez display: none. Et enfin, parmis celles que je trouve le plus utile, il y a display: flex.

Le style de display flex va transformer votre élément en flexbox. Je ne vais pas forcément expliquer tout le fonctionnement des flexboxs ici, parce que déjà ça serait long, et ensuite j'ai pas encore compris toutes les subtilitées.Donc comme d'habitude vous retrouverez dans la description un lien vers la documentation de Mozilla qui est très bien fichus.

Bon! Les flexboxs servent à agencer vos pages de manière flexibles. Avant quand on voulait mettre par exemple plusieurs éléments sur la même ligne on utilisait des tableaux. Maintenant on utilise flexbox. Vous allez pouvoir définir comment les enfants d'un élément

doivent s’afficher, en colonne ou en ligne par exemple, changer leur ordre, comment il vont devoir prendre plus ou moins de place ou enfin comment vous voulez gérer leur alignement vertical. Bref c’est vraiment un outil extrêmement puissant mais très complexe. Dans la description je vous mettrais aussi un lien vers le jeu Flexbox Froggy qui vous explique très bien les bases des flexbox.

Et voilà c’est là que je m'arrêterais sur les fonctionnalités du CSS. Comme vous l’aurez compris le langage est très important et très complet mais il serait trop long de tout expliquer.

Mais vous me direz “et si j’ai pas envie de faire de CSS je fais comment”, et bien il existe beaucoup de feuille de style toutes faites, qui vous fournissent tout ce dont vous avez besoin. Par exemple il y a Tailwind CSS qui inclut directement tout un tas de classes à ajouter pour styliser directement vos pages, ou encore Bulma qui fait plus ou moins la même chose. J’ajouterais aussi deux autres template que je trouve cool, NES.css qui donnera directement un style retro pixel à vos applications, ou encore 98.css qui lui donnera un aspect Windows 98!

Voilà pour le CSS classique. Pour conclure cet épisode, je vais tout de même vous parler des améliorations qui commencent à arriver pour le CSS. En effet sur certains points, le langage a des manques et c’est là qu’arrivent des langages comme LESS ou SASS. En gros vous allez écrire vos feuilles dans ces langages (qui s’approchent tout de même énormément du CSS mais ajoutent quelques idées aux syntaxes), et soit vous convertissez ce code en CSS avant de publier le site, soit il sera converti dynamiquement grâce à du JavaScript. Ils sont très puissant et peuvent vraiment vous simplifier la vie, en ajoutant par exemple des variables améliorées, des conditionnelles, des répétitions ou des fonctions utilitaires… Des choses dont manque le CSS.

Et je pense que on a fait le tour globalement des Cascade StyleSheet. Avant de terminer je souhaitait quand même rappeler que le CSS est un langage qui évolue et chaque attribut n’est pas forcément compatible avec chaque navigateur. A vous de voir si vous voulez les utiliser tout de même, au risque de vous couper d’une partie des utilisateurs, ou alors de partir sur des balises compatible partout, en vous coupant de certaines des dernières capacitées. Mais rassurez vous, les dernières version de Chrome et de Firefox sont souvent sur la même longueur d’onde.

Voilà, j’espère que cet épisode vous aura plus. Désolé de l’attente mais les cours et le manque de motivation auront un peu pris le pas sur les podcasts. Du coup à partir des prochain épisode, je vais élargir un peu les contours de l’émission, pour parler de technologie informatique et non plus uniquement de langages. Je pense que ça sera beaucoup liés aux technologies du Web parce que c’est celles qui m'intéressent le plus, mais je n’exclue pas de parler d’autres choses. La prochaine fois on va donc parler du JavaScript puisque c’est un langage super cool, mais ensuite on parlera sécurité informatique avec les Json Web Token!

Je voulais aussi remercier les personnes qui m'ont aidées en relisant cet épisode pour voir si je ne dis pas n'importe quoi, à savoir Mankoty et Pof. Comme d'habitude, n'hésitez pas à vous abonner au podcast, sur presque toutes les plateformes, et sur Youtube aussi. Pour me soutenir financièrement c'est sur utip.io/bigaston, pour me suivre c'est sur Twitter @Bigaston. Je vous dis à la prochaine. Tchao!

https://fr.wikipedia.org/wiki/Feuilles_de_style_en_cascade
https://developer.mozilla.org/fr/docs/Web/CSS
https://caniuse.com/
https://developer.mozilla.org/fr/docs/Web/CSS/Pseudo-%C3%A9l%C3%A9ments
https://developer.mozilla.org/fr/docs/Web/CSS/Pseudo-classes
https://developer.mozilla.org/fr/docs/Web/CSS/S%C3%A9lecteurs_CSS#Les_combinateurs
https://developer.mozilla.org/fr/docs/Web/CSS/CSS_Flexible_Box_Layout/Concepts_de_base_flexbox
https://flexboxfroggy.com/#fr
https://tailwindcss.com/
https://bulma.io/
https://nostalgic-css.github.io/NES.css/
https://jdan.github.io/98.css/
http://lesscss.org/
https://sass-lang.com/