

📍 Avenue V. Maistriau 8a
B-7000 Mons
📞 +32 (0)65 33 81 54
✉️ scitech-mons@heh.be

WWW.HEH.BE



UE : Développement web

- AA : Développement web – théorie
Ivan Miller
- AA : Développement web – travaux pratiques
Joan Claus & Joakim Chapelle

Bachelier en Informatique et systèmes
orientation réseaux et télécommunications

Index

Bienvenue.....	- 5 -
Intro : tour d'horizon du développement web	- 6 -
Devenir webmaster.....	- 6 -
Résumé des principales technologies web	- 6 -
Histoire des navigateurs.....	- 9 -
Chapitre 1 : HTML	- 15 -
1.1 Introduction au HTML.....	- 15 -
Rôle du langage HTML.....	- 15 -
Non-maîtrise de l'affichage	- 15 -
Programmes nécessaires	- 16 -
1.2 Les bases du langage HTML	- 18 -
Les éléments	- 18 -
Classification du langage HTML.....	- 18 -
Les attributs	- 19 -
Les balises auto-fermantes	- 20 -
Les commentaires.....	- 20 -
La page HTML	- 21 -
Traitement des espacements par le navigateur.....	- 22 -
Le flux.....	- 22 -
L'arborecence	- 23 -
1.3 Les formats d'encodage	- 24 -
De l'ASCII à l'UTF-8	- 24 -
Les entités de caractères	- 26 -
1.4 Les familles d'éléments	- 27 -
Les éléments lignes.....	- 27 -
Les éléments blocs.....	- 27 -
Nouvelles catégories HTML5	- 28 -
1.5 Les principaux éléments HTML	- 29 -
Les titres et les paragraphes.....	- 29 -
Les éléments d'enrichissement du texte.....	- 30 -
Les éléments de structure de la page.....	- 31 -
Les éléments interdits	- 31 -
Quelques éléments spéciaux	- 32 -
1.6 Les couleurs.....	- 33 -
Coder des couleurs par leurs noms	- 33 -
Coder des couleurs par code hexadécimal	- 33 -
Noms de couleurs	- 34 -
1.7 Les listes	- 36 -
Listes ordonnées et non-ordonnées	- 36 -
Listes de définitions ou d'associations	- 37 -
Exercice : Menu du jour	- 38 -
1.8 Les tableaux	- 39 -
Exercice : Pizza ou choucroute	- 41 -
Exercice : Animaux de la forêt	- 42 -
1.9 Les liens	- 43 -
L'hypertexte.....	- 43 -
Les liens.....	- 43 -
Envoi d'email via un lien	- 43 -
Les protocoles HTTP et HTTPS	- 44 -

Erreurs	- 44 -
Les URL.....	- 44 -
Les URL absolues.....	- 45 -
Les URL relatives.....	- 45 -
Les ancrés	- 45 -
1.10 Les images	- 46 -
Les formats d'images adaptés au Web.....	- 46 -
Les résolutions d'image sur le Web	- 47 -
Les images :	- 48 -
Exercice : les URL et les images	- 49 -
Légendes de médias	- 50 -
Images adaptatives.....	- 50 -
Les cartes	- 51 -
Les icônes avec <i>Font Awesome</i>	- 52 -
La <i>Favicon</i>	- 52 -
Les sons et les vidéos.....	- 53 -
1.11 Les frame et iframe	- 54 -
1.12 Les formulaires.....	- 55 -
Formulaire : <form>.....	- 55 -
Conteneur : <fieldset>	- 56 -
Champs de saisie : <input>	- 56 -
Zone cliquable étendue : <label>.....	- 58 -
Champ multi-lignes : <textarea>	- 58 -
Listes déroulantes : <select>	- 59 -
Liste d'autocomplétion : <datalist>.....	- 59 -
Fichiers joints.....	- 60 -
Champ résultat : <output>	- 60 -
Exercice : formulaire	- 61 -
1.13 Les balises meta	- 62 -
Exercice : Analyse de méta-balises	- 63 -
Exercice final HTML : trouvez les erreurs W3C.....	- 64 -
Chapitre 2 : CSS	- 66 -
2.1 Introduction aux feuilles de style CSS	- 66 -
2.2 Comment utiliser les CSS.....	- 67 -
3 différents types de feuilles de style.....	- 67 -
Comment écrire une feuille de style	- 68 -
C comme Cascade	- 68 -
Commentaires en CSS.....	- 68 -
Les médias	- 69 -
2.3 Les sélecteurs	- 70 -
Les sélecteurs CSS.....	- 70 -
Récapitulatif des sélecteurs CSS	- 74 -
Exercice : les sélecteurs CSS	- 75 -
Exercice : rédigez des règles CSS	- 75 -
2.4 Les déclarations et leurs priorités.....	- 76 -
Déclarations, héritage et cascade.....	- 76 -
Les règles CSS du navigateur	- 76 -
Imposer une déclaration prioritaire	- 77 -
Priorité des déclarations.....	- 77 -
Exercice : priorité des sélecteurs	- 79 -
Exercice : priorité des sélecteurs	- 80 -
2.5 Les unités de mesure	- 81 -

Les dimensions.....	- 81 -
Les dimensions pour l'impression	- 81 -
Les cadratins	- 81 -
Les angles.....	- 82 -
Le temps et les fréquences.....	- 82 -
<i>calc()</i>	- 82 -
Variables CSS.....	- 82 -
2.6 Les propriétés.....	- 83 -
Couleur	- 83 -
Opacité.....	- 83 -
Police de caractères.....	- 84 -
Mise en forme des textes	- 85 -
Mise en forme des textes – propriétés peu utilisées	- 87 -
Curseur.....	- 87 -
Colonnes multiples	- 88 -
Largeur et Hauteur	- 88 -
Bordures et marges	- 89 -
Calcul des dimensions des blocs.....	- 90 -
Alignment : centrer un élément	- 90 -
Listes à puces	- 90 -
Arrière-plan.....	- 91 -
Arrière-plan – dégradés	- 91 -
Arrière-plan – propriétés peu utilisées.....	- 92 -
Guillemets.....	- 92 -
Saut de page (pour le <i>print</i>)	- 92 -
Ce qui dépasse	- 93 -
Rognures.....	- 93 -
Transitions	- 93 -
Animations.....	- 94 -
Transformations	- 95 -
<i>Scroll</i>	- 95 -
Filtres	- 96 -
Valeurs universelles	- 96 -
2.7 Les propriétés de positionnement.....	- 97 -
Le flux.....	- 99 -
La position relative (<i>position:relative</i>)	- 100 -
Les flottants	- 100 -
La position absolue (<i>position:absolute</i>).....	- 102 -
La position fixe (<i>position:fixed</i>).....	- 103 -
La position <i>sticky</i> (<i>position:sticky</i>)	- 103 -
Les écarts : <i>top</i> , <i>bottom</i> , <i>left</i> et <i>right</i>	- 104 -
L'ordre de superposition des éléments : <i>z-index</i>	- 104 -
FlexBox.....	- 105 -
Exercice : Navigation en Flexbox	- 108 -
Grid Layout	- 110 -
Exercice : <i>Fandom</i> en Grid Layout	- 113 -
2.8 Responsive Design.....	- 115 -
Statique, fluide, adaptatif ou <i>responsive</i> ?	- 115 -
Les définitions d'écran.....	- 115 -
Le viewport	- 116 -
Les media-queries.....	- 116 -
Les solutions pratiques pour un comportement <i>Responsive</i>	- 116 -

2.9 L'accessibilité du Web.....	- 118 -
Petit historique et législation actuelle	- 118 -
L'accessibilité	- 118 -
Les handicaps concernés	- 119 -
Résumé des critères d'accessibilité	- 120 -
2.10 Les problèmes de compatibilité et leurs solutions	- 121 -
Les préfixes CSS	- 121 -
Les commentaires conditionnels	- 122 -
Les hacks CSS	- 122 -
2.11 Quelques outils CSS.....	- 123 -
2.12 Exemples HTML+CSS	- 124 -
Exemple : un contenu qui chevauche un autre	- 124 -
Exemple : une disposition texte + image en 50/50	- 125 -
Exemple : des grands pavés de navigation (image + texte).....	- 126 -
Exemple : la technique des <i>sprites</i>	- 127 -
Exemple : personnaliser les boutons <i>radio</i> ou <i>checkbox</i>	- 128 -
Exemple : animation d'élément au chargement de la page.....	- 129 -
Exemple : navigation déroulante.....	- 130 -
Chapitre 3 : Ergonomie des sites web	- 2 -
3.1 L'ergonomie	- 2 -
UI & UX design	- 3 -
Les maquettes : zonings, wireframes, layouts, mockups	- 4 -
3.2 Utilité et utilisabilité.....	- 5 -
Efficacité	- 6 -
Efficience.....	- 7 -
Satisfaction	- 8 -
3.3 Quelques (mauvaises) idées reçues sur l'ergonomie web.....	- 10 -
3.4 Lois de Gestalt.....	- 12 -
La proximité	- 12 -
La familiarité	- 13 -
La similarité.....	- 13 -
3.5 Autres principes fondamentaux de l'ergonomie	- 14 -
La loi de Fitts	- 14 -
L'affordance	- 15 -
Le nombre magique de Miller	- 16 -
La lisibilité	- 16 -
Chapitre 4 : Référencement web	- 18 -
Les débuts du référencement	- 18 -
Les algorithmes de référencement.....	- 18 -
Le référencement et le positionnement	- 18 -
La page de résultats (SERP)	- 19 -
Le référencement naturel (SEO) et sponsorisé (SEA)	- 20 -
À propos de Google.....	- 20 -
Punitions Google	- 21 -
Mots-clés et <i>spiders</i>	- 21 -
Liens internes et liens entrants.....	- 21 -
Attaques SEO.....	- 22 -
<i>Duplicate Content</i>	- 22 -
<i>Sitemaps</i>	- 22 -
En conclusion, comment améliorer le référencement ?	- 23 -
Projet de site web.....	- 24 -
Sources	- 26 -

Bienvenue

Bonjour,



Bienvenue au cours de **Développement web** dont l'objectif est de vous apprendre les bases de la création de sites web.

En guise d'introduction, nous effectuerons un tour d'horizon des technologies web : les langages, les outils, l'historique du Web, les navigateurs, etc.

Nous aborderons principalement le développement de sites web statiques avec les deux premiers langages informatiques du web : **HTML** pour la structure des sites web et **CSS** pour leur mise en page. Nous en profiterons pour aborder les normes du W3C, les problèmes de compatibilité, le *responsive design* et l'accessibilité web.

Nous verrons ensuite comment appliquer des règles d'**ergonomie** aux sites web afin de les rendre simples d'utilisation, intuitifs et attractifs. À l'aide d'une série de bons et de mauvais exemples, nous apprendrons à optimiser nos interfaces web et à les adapter à leur public.

Pour terminer, nous aborderons le **référencement web** : les bases pour positionner vos sites dans les résultats des moteurs de recherche.

Ce cours est soutenu par des outils en ligne :

- Sur l'**eCampus** (<https://ecampus.heh.be>), vous trouverez le syllabus en pdf, des fichiers nécessaires aux exercices, des exercices supplémentaires, les fichiers des démonstrations faites en classe, etc. C'est également par cette plateforme que se fait la remise du projet de fin d'année.
- Sur **Quizineur** (<https://www.heh.be/quizineur>), vous pourrez tester vos connaissances en vous exerçant sur des QCM aléatoires portant sur l'ensemble de la matière et provenant d'une base de données de plus de 500 questions aux paramètres variables.

Vous rencontrerez 2 pictogrammes en parcourant ce syllabus :



Ce pictogramme signale des informations complémentaires, pour les plus téméraires d'entre vous. Les chances d'être interrogé sur cette matière sont très faibles, mais vous pourriez en avoir besoin lors d'exercices ou de projets personnels.



Ce pictogramme annonce des critères d'accessibilité appelés WCAG : un ensemble de normes assurant l'accès aux sites web à tout un chacun, y compris les personnes en situation de handicap.

Ivan Miller
ivan.miller@heh.be

Intro : tour d'horizon du développement web

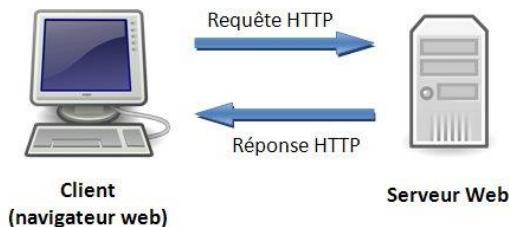
Devenir webmaster

Lors de ses débuts en 1990, le Web se limitait à un semblant de téletexte bas de gamme, sans image, sans dynamisme et, selon certains, sans avenir. 30 ans plus tard, le Web compte plus d'1.7 milliard de sites et la moitié de la population mondiale utilise ce super-moyen de communication, devenu indispensable à chaque entreprise, à chaque commerce et à chaque personne.

Pratiquer le développement web nécessite de nos jours l'apprentissage d'un grand nombre de technologies et de langages : les langages HTML, CSS, JavaScript, PHP, XML, les bases de données, les navigateurs, les normes du W3C, les normes d'accessibilité WCAG, l'ergonomie web, le référencement, le *responsive design*, les solutions d'hébergement, etc. sont autant de savoirs nécessaires aux webmasters et à la profession de développeur web.

Résumé des principales technologies web

Un site web est un ensemble de fichiers informatiques stockés sur un serveur. À la demande, c'est-à-dire lors de la réception d'une requête en provenance du visiteur (client), le serveur exécute certains scripts, puis envoie les fichiers au client. Les fichiers reçus sont interprétés par le navigateur de la machine du visiteur.



Il y a donc lieu de faire cette distinction entre les technologies qui opèrent chez le client, appelées technologies ***front-end***, et les technologies opérant sur le serveur, appelées ***back-end***. Ce cours porte sur des technologies *front-end* : l'HTML et le CSS.

1. Front-end (côté client)

HTML : l'HTML (*HyperText Markup Language*) est le premier langage du Web et sert à coder la structure des pages. Il s'agit d'un langage de description, balisé et interprétré.

Le langage HTML a fortement évolué depuis sa création, passant de la version HTML (± 1990), à la version HTML 2 (1995), puis HTML 3 (± 1996), suivie d'HTML 4 (1997), pour évoluer ensuite en XHTML 1.0 (2000) et déboucher sur l'**HTML5** actuel (2009), prévue pour être la dernière version, en constante évolution.



XHTML : évolution du langage HTML4 adaptée aux normes du XML et donc plus stricte que les autres versions HTML. Il subsiste encore de nombreux sites codés en XHTML actuellement, mais ils sont progressivement remplacés par de l'HTML5.

CSS : en 1996, apparaît le langage CSS (*Cascading Style Sheet*) qui permet de regrouper toutes les informations de mise en page (couleurs, polices, dimensions, etc.) dans un seul fichier. Le CSS réduit le poids des sites, garantit l'homogénéité de leur mise en page, facilite leur maintenance et leurs mises à

jour, permet de nouvelles options de mise en page, etc. Le langage CSS en est actuellement à sa version 3, dont le support par les navigateurs n'est que partiel, mais en progression constante (voir <http://css3test.com/>).

Les pages web comportant uniquement les langages CSS et HTML sont dites « **statiques** », en opposition aux pages web **dynamiques**, générées par un langage dynamique côté serveur et dont le contenu est variable.

W3C : La promotion de la compatibilité des technologies web HTML5, HTML, XHTML, XML, XSL, CSS, PNG et SVG est confiée au **W3C** (*World Wide Web Consortium*). Le W3C met à disposition un outil gratuit en ligne appelé le Validator et servant à tester la validité de vos codes HTML (ainsi qu'un Validator CSS). Depuis sa création, le W3C est dirigé par Tim Berners-Lee.

Flash (langage **ActionScript**) : ancien logiciel d'animation vectorielle, propriété de Macromedia, puis racheté par Adobe en 2005. Le Flash permet la création d'animations graphiques de qualité mais comporte certains désavantages : payant, lourd, nécessitant l'installation de plugins et la connaissance du langage ActionScript. Flash a connu un franc succès pendant une vingtaine d'années avant de péricliter.

JavaScript : en 1996, la même année que le CSS, apparaît le JavaScript : langage de programmation objet pour le web. Avec le PHP, il s'agit des deux langages de programmation dynamiques pour le web les plus répandus. Contrairement au PHP, le JavaScript est interprété côté client, par le navigateur. Le JavaScript est également un langage événementiel : il permet de détecter les actions des visiteurs (le clic, le survol, le *focus*, le *scroll*, etc.) et d'y attribuer des actions.

jQuery : bibliothèque JavaScript permettant la création d'effets graphiques avancés (mouvements, fondus, etc.) et garantissant une meilleure compatibilité, jQuery est le framework front-end le plus répandu grâce à ses nombreux avantages : gratuité, légèreté, simplicité.

AJAX : en 2005, le JavaScript s'octroie une méthode de programmation alliant JavaScript et XML (*Asynchronous Javascript And Xml*) qui va permettre aux sites de charger des données sans recharger la page. Exemple : Google Maps qui charge des portions de cartes suivant les actions du visiteur sans recharger l'ensemble de la page.

Node.js : en 2009, Ryan Dahl lance une plateforme logicielle libre en JavaScript : du JavaScript côté serveur améliorant entre autres les échanges serveur-client.

JSON : créé par Douglas Crockford entre 2002 et 2005, JSON (*JavaScript Object Notation*) est un format léger d'échange de données sous forme textuelle permettant de structurer les informations.

Exemple : { "album": "Sehnsucht", "groupe": "Rammstein", "annee": 1997 }

2. Back-end (côté serveur)

PHP : *Personal Home Page Hypertext Preprocessor*. Le PHP est un langage dynamique qui a la particularité d'être interprété côté serveur (et non côté client comme l'HTML, le CSS ou le JavaScript), ce qui garantit la sécurité du code. Le PHP est également un langage nativement développé pour la communication avec les bases de données MySQL. Bien que les chiffres diffèrent d'une source à l'autre, on peut estimer qu'environ 80% des sites web utilisent le PHP.



ASP.NET : successeur de l'ASP, l'ASP.NET est un ensemble de technologies web propriétaires (Microsoft). Environ 20% des sites web utilisent l'ASP.NET ou son ancêtre ASP.

Parmi les autres langages côté serveur, on retrouve **Java**, **ColdFusion**, **Node.js**, **Perl**, **Ruby** et **Python**.

XML : langage balisé pour stocker des données de manière semi-structurée. Ses règles de syntaxe sont très proches de celles de l'HTML et ses principaux avantages sont sa légèreté et sa transportabilité.

MySQL : Système de gestion de bases de données relationnelles (parmi les autres SGBDR, citons **DB2**, **Microsoft SQL Server**, **SQLite**, **Oracle Database**, **PostgreSQL**, **Ingres**). Cette solution est la plus performante et la plus utilisée pour le stockage de données mais est plus lourde à mettre en place qu'un simple fichier XML. En 2008, MySQL est racheté par Sun Microsystems et depuis 2009, le développement Open Source du projet continue sous le nom de **MariaDB**.

SQL : *Structured Query Language*, le langage de requête utilisé dans la plupart des SGBDR.

3. Les Systèmes de gestion de contenu

CMS : Les **Content Management System** ou système de gestion de contenu sont des logiciels permettant de créer des sites destinés à gérer de grandes quantités de données. Ils ont pour particularité de permettre cette création sans pour autant devoir programmer. Parmi les plus répandus on retrouve **Joomla!**, **Drupal**, **SPIP** et **Plone** ; et dans la catégorie e-commerce, citons **Magento**, **Prestashop** et **Shopify**.

En arrière-plan, les CMS sont le plus souvent codés en PHP, possèdent une base de données MySQL, utilisent le JavaScript, le CSS et l' HTML et respectent les normes du W3C. Bref, les technologies figurant au programme de votre cursus de Bachelier à la HEH.

L'utilisation des CMS permet une productivité accrue en simplifiant le travail du développeur mais n'est pas idéale à l'élaboration de solutions sur mesure.

Logiciels de création de blogs : Il est toléré de confondre les CMS avec les logiciels de création de blogs comme **WordPress**, **Dotclear**, **Typo** ou **Movable Type**. La différence entre ces deux catégories est bien mince car bon nombre de ces logiciels de création de blogs offrent de plus en plus de fonctionnalités, devenant ainsi des CMS spécialisés dans la création de blogs.

Parmi toutes ces solutions, **WordPress** est largement la plus répandue avec 27% des sites web dans le monde, devant les 3% de Joomla! et les 2% de Drupal.

Histoire des navigateurs

Les navigateurs sont les logiciels qui permettent de visualiser des contenus en ligne. De nos jours et dans le monde entier, ce sont les logiciels les plus utilisés.

De la création du premier navigateur à l'aventure Netscape et l'arrivée foudroyante de Microsoft Internet Explorer, puis au succès de Google Chrome, le monde des navigateurs a beaucoup évolué depuis 1990. Le Web aussi.



 Le premier navigateur est créé en 1990 par Tim Berners-Lee lorsque celui-ci pose les bases du futur Web mondial. Baptisé **World Wide Web**, ce navigateur fonctionne alors sur la plateforme NeXT et propose un nombre important de fonctionnalités dont un module d'édition wysiwyg permettant l'ajout d'informations. Toutefois, les plateformes NeXT n'étaient pas très répandues et il devient important de développer une version plus ouverte au public.

Au mois de février 1993, alors que le Web ne comptait que 200 sites, une équipe de chercheurs du NCSA de l'université de l'Illinois propose **Mosaic**, premier navigateur graphique. Mosaic permet pour la première fois l'affichage des images (GIF) et des formulaires, donnant ainsi un nouveau souffle au monde des navigateurs et par conséquent, au Web.



 En 1994, Mosaic évolue vers une société et un logiciel du même nom : **Netscape Navigator**. C'est le premier navigateur à supporter les tableaux HTML et les feuilles de style CSS. Le succès est immédiat. En téléchargement libre sur internet, distribué gratuitement aux universités et aux chercheurs, Netscape Navigator devient le navigateur le plus utilisé avec presque 90% de part de marché, malgré de nombreux concurrents comme IBM webExplorer, CyberDog ou Lynx.

En 1994, Bill Gates aurait déclaré « le Web, ça ne marchera jamais. » Cette citation n'est pas avérée, mais l'intéressé reconnaît être passé à côté de cette immense opportunité, pendant un temps. Heureusement pour Bill, un de ses assistants, Steven Sinovksy, le convainct que les étudiants dans les universités sont friands de navigateurs web. Un projet mineur de création de navigateur est alors mené chez Microsoft.



C'est en 1995 que Bill Gates a le déclic et se rend compte à quel point l'avenir de l'informatique se passe sur le Web. Il l'annonce d'ailleurs à ses lieutenants dans un mémo devenu célèbre : « The Internet Tidal Wave ». Microsoft change alors de politique et met la priorité sur le développement de **Microsoft Internet Explorer**. Ce navigateur gratuit est installé par défaut avec le système d'exploitation Windows 95, ce qui fait l'objet d'un procès d'envergure pour abus de position dominante. Le procès se termine en 2004 sur une amende record de 497,2 millions d'euros et plusieurs obligations à respecter.

Peu à peu, le retard technologique d'Internet Explorer sur Netscape est rattrapé par l'équipe de développement de Microsoft menée par Brad Silverberg et la position dominante de Netscape est mise à mal.

Netscape réagit à cette nouvelle menace du marché. On assiste alors à la multiplication de **mécanismes propriétaires** durant ce qu'on appelle aujourd'hui la « **guerre des navigateurs** ». Les deux sociétés tentent de s'approprier le langage HTML en imposant des normes réservées à leur navigateur, comme la balise *<blink>* qui ne fonctionne que sur Netscape, ou la balise *<marquee>* sur Explorer. Les incompatibilités se multiplient. Deux types de Web se dessinent et les développeurs éprouvent de grandes difficultés à créer des sites réellement compatibles sur les deux navigateurs concurrents.

Au mois d'août 1996, à seulement 3 jours d'écart, Netscape et Microsoft lancent simultanément la version 3 de leur navigateur. Si cette nouvelle version ne ressemble qu'à une simple mise à jour pour Netscape, elle constitue en revanche une avancée majeure pour IE qui supporte dorénavant le CSS, le JavaScript et la technologie ActiveX.

La gratuité d'Internet Explorer, sa rapide évolution technologique, son installation comprise avec le système d'exploitation Windows, et l'appui d'AOL (société américaine de services internet) aident le navigateur de Microsoft à grimper en flèche.

Comme IE connaît un succès croissant, de plus en plus de sites web sont développés pour ce navigateur au grand malheur du W3C puisqu'à l'époque, IE ne respecte pas les standards HTML et CSS. Ceci implique que la plupart des sites de l'époque sont codés de manière inexacte et incompatible avec les autres navigateurs.

Le **W3C** cherche tant bien que mal à ramener un peu d'ordre en tentant d'établir des standards obligatoires.

En 1998, coup de théâtre, l'entreprise Netscape lance la **fondation Mozilla** et diffuse gratuitement le code source de son navigateur sur le site mozilla.org, ceci pour associer des talents extérieurs au développement du logiciel. Le nom de cette fondation à but non lucratif, Mozilla, était le nom de code du développement du navigateur Netscape, destiné à être le « Mosaic Killer ». Le nom Mozilla provient de la contraction de « Mosaic Killer » et du jeu de mot avec « Godzilla ». Mozilla est ensuite devenu, sous la forme d'un lézard vert, la mascotte des développeurs de Netscape Communications Corporation.



Vers un monopole d'Internet Explorer : Malgré tout, le fossé s'élargit entre les deux concurrents. Et si Netscape continue à optimiser son outil (avec Netscape 4.7), MSIE 5.0 apparaît en 1999 avec un nombre important d'améliorations dont le support du langage CSS. MSIE est alors le premier navigateur utilisé sur le Web avec plus de 50% de parts de marché. Le temps pas si lointain où Netscape occupait presque 90% du marché semble bien révolu.

Le mois de janvier 2000 consacre enfin la victoire de MSIE sur Netscape. Le premier possède 75% des parts de marché et le second, uniquement 20%. Et les autres ? Seul Opéra, lancé en 1996, parvient tant bien que mal à se créer une place avec 1% de parts de marché.

La fin de Netscape : En 1999, AOL rachète Netscape pour 4 milliards de dollars. Après la sortie de la version 9 en 2007, AOL décide de stopper le développement du navigateur et fait un don de 2 millions de dollars à la fondation Mozilla.



En 2002, après un long travail de développement sort la première version du navigateur **Firefox**, nommé alors *Phoenix*, puis renommé *Firebird* en 2003, puis *Firefox* en 2004. En anglais, le *Firefox* désigne le panda roux, mais l'équipe de graphisme travaillant sur le logo a préféré styliser un renard enflammé.

Internet Explorer domine largement le marché mondial des navigateurs et repose sur ses lauriers. Microsoft fait passer le développement d'IE au second plan, pour se concentrer sur leur système d'exploitation. Le navigateur de Microsoft prend à nouveau du retard, et accumule les problèmes de compatibilité : la non-consistance des blocs flottants, la transparence des PNG, le calcul des dimensions des blocs, du retard dans le support de CSS 2, etc.



Ces dysfonctionnements finissent par coûter cher à IE et de plus en plus de personnes optent pour le navigateur Firefox : gratuit, puissant et convivial.



En 2003, Apple réagit à la stagnation d'Internet Explorer et développe son propre navigateur : **Safari**. Safari est actuellement le 2^e navigateur le plus utilisé au monde, derrière Chrome. Safari n'est plus compatible Windows depuis la version 5.1.7 de 2012.

En 2004, le navigateur pour mobile **UC browser** voit le jour. Disponible sur Android, iOS, Windows Phone, Symbian et Blackberry, UC browser trouve surtout son public dans les pays asiatiques.



En 2008, alors que Firefox est sur le point de détrôner IE, Google lance son nouveau navigateur : **Chrome**. Celui-ci propose une interface très épurée en rassemblant les barres d'outils et les menus en seulement 2 icônes. Mais la grande nouveauté est la fusion entre la barre d'adresse et le champ de recherche, formant une barre intelligente. Chrome détrône rapidement tous ses concurrents devenant ainsi le logiciel le plus utilisé au monde !

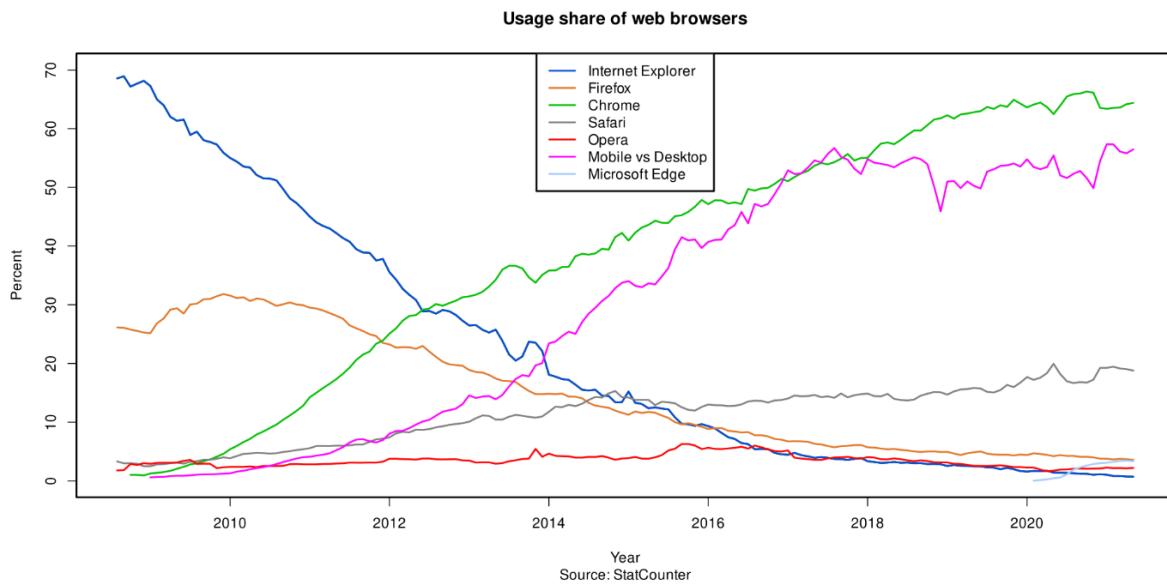
En 2010, le projet XHTML 2 est abandonné, l'accessibilité devient obligatoire dans plusieurs pays et le W3C arrive enfin à imposer ses standards aux différents navigateurs du marché.

Depuis 1999 et le Nokia 7110, l'accès au Web depuis un mobile est rendu possible, mais ce n'est qu'en 2010 avec l'essor des Smartphones que le **Web mobile** connaît une réelle croissance, dépassant les 30% d'utilisation en Belgique, 40% au niveau européen et plus de 50% au niveau mondial ! Alors que la problématique de l'adaptation des sites aux différentes définitions d'écran est vieille comme le Web, les développeurs sont dorénavant confrontés à des définitions minuscules. C'est l'aire du Responsive Design.



En 2015, Microsoft lance son nouveau navigateur, **Edge**, moderne, conforme aux standards et rapide. C'est dorénavant Safari le dernier de classe en matière de support des nouvelles technologies...

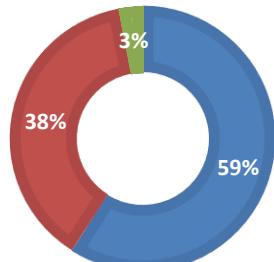
Aujourd'hui, la course entre navigateurs bat son plein : chaque navigateur se doit d'être convivial, puissant et compatible. Les développeurs sont moins confrontés aux problèmes d'incompatibilités des navigateurs et peuvent dorénavant se concentrer sur les problèmes de support des nouveautés HTML5 et CSS3, et sur les difficultés d'adaptation des sites sur mobiles.



Le graphique ci-dessus représente les principales tendances au niveau mondial, depuis 2008 : le triomphe de Chrome, le succès de Safari, la relative stabilité d'Opera, le retrait de Firefox, la débâcle d'Internet Explorer, l'arrivée récente d'Edge, et, en rose, la tendance à surfer depuis un mobile.

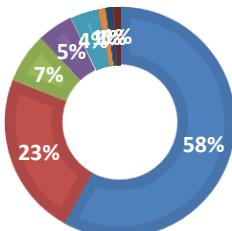
CATÉGORIES D'APPAREILS EN BELGIQUE, 2021

■ PC ■ Mobiles ■ Tablettes

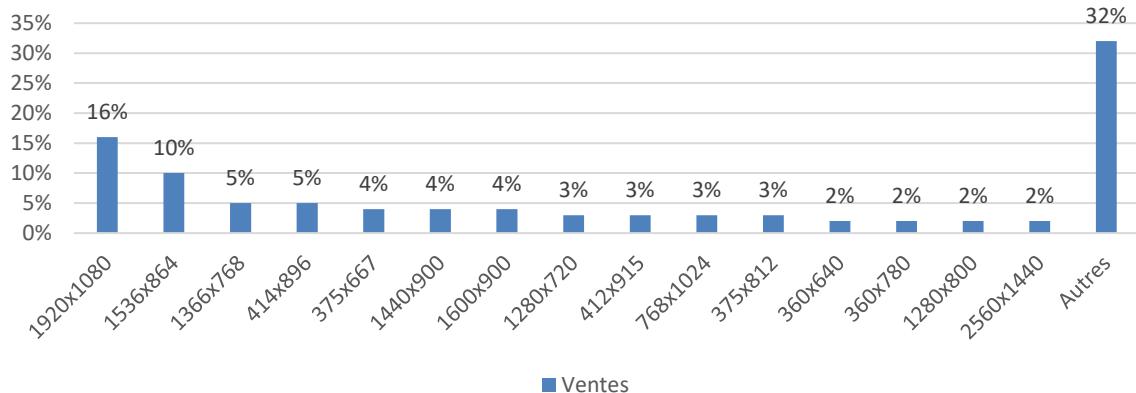


NAVIGATEURS EN BELGIQUE, 2021

■ Chrome ■ Safari ■ Edge ■ Firefox
■ Samsung ■ Opera ■ IE ■ Autres

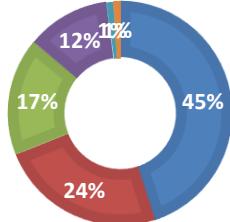


Définitions d'écran en Belgique, 2021



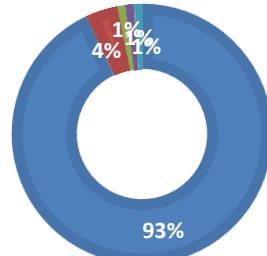
SYSTÈMES D'EXPLOITATION EN BELGIQUE, 2021

■ Windows ■ Android ■ iOS ■ OS X ■ Linux ■ Autres



MOTEURS DE RECHERCHE EN BELGIQUE, 2021

■ Google ■ Bing ■ Ecosia ■ Yahoo! ■ Autres



Source : <http://gs.statcounter.com/>

Chapitre 1 : HTML



Chapitre 1 : HTML

1.1 Introduction au HTML

Rôle du langage HTML

HTML est le langage universel de structure des informations sur le Web.

Les fichiers HTML contiennent les textes, enrichis d'instructions HTML apportant de la sémantique, c'est-à-dire des précisions sur le rôle des parties de texte (titres, paragraphes, adresses, citations, etc.). Les instructions HTML peuvent aussi apporter des fonctionnalités (comme un formulaire, un lien, une image, une vidéo, etc.), ou servir de conteneur aux différentes parties de la page (un en-tête, une marge, une zone de navigation, un pied de page, etc.).

Au HTML, viennent se greffer des instructions CSS définissant la mise en page du site : le CSS permet de modifier les couleurs, les polices, les tailles et tous les paramètres de présentation visuelle.

Par exemple, afin de coder une liste de courses sous forme de page web, c'est en **HTML** que vous indiquez la présence d'un titre « courses » et d'une liste d'aliments « pommes, carottes, patates, etc. », mais c'est en **CSS** que vous définissez la couleur et la police du texte.

Aux débuts du Web, le CSS n'existait pas. C'est pourquoi HTML est un langage comprenant quelques anciennes instructions de mise en page. Bien que ces éléments de mise en page HTML soient brièvement abordés dans ce cours, nous veillerons à ne pas les utiliser et à toujours **séparer la structure de la présentation** lors du développement de nos sites web.

Non-maîtrise de l'affichage

En matière de développement web, une des difficultés auxquelles vous êtes confrontés est la diversité des terminaux. L'affichage de vos sites web varie selon le périphérique (mobile, tablette, console ou desktop), le système d'exploitation, les polices disponibles, le navigateur et sa version, la définition de l'écran (souvent appelée à tort « résolution » d'écran) : autant de paramètres influençant l'affichage des sites.

Dès lors, il est inenvisageable de viser un affichage parfaitement identique sur tous les terminaux. L'essentiel est d'assurer **l'accessibilité** des contenus : tout le monde doit pouvoir prendre connaissance des informations reprises dans vos sites.



Programmes nécessaires

Pour développer une page HTML, deux solutions s'offrent à vous :

La solution assistée : les éditeurs WYSIWYG (What You See Is What You Get), ou en français les éditeurs « tel-tel » (tel écran, tel écrit), sont des éditeurs présentant le résultat de la page web et permettant d'interagir avec celui-ci afin de le personnaliser. Certains de ces éditeurs permettent aussi la modification du code.

Parmi les plus célèbres, citons FrontPage, Adobe Dreamweaver, Expression Web Designer, Adobe GoLive, Nvu, Claris Home Page, WebExpert, Netscape Editor, etc. Ces éditeurs, appréciés pour leur facilité d'utilisation et l'observation du résultat en temps réel, ne sont pas adaptés à l'apprentissage du développement web et ne permettent pas autant de souplesse que la seconde solution...

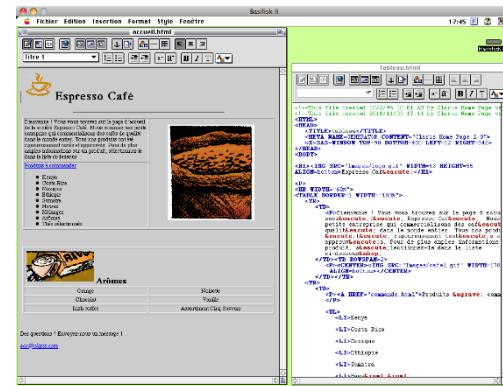


Figure 1 : Claris Home Page, un des premiers WYSIWYG

La solution des professionnels : les éditeurs de code tels que Notepad++, Sublime Text, Smultron, Visual Studio Code, Aptana, Vim, Atom, UltraEdit, Brackets, etc.

Moins conviviaux de premier abord, leur utilisation permet de tout coder à condition de connaître les langages informatiques.

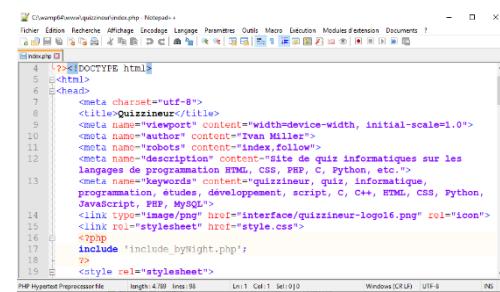


Figure 2 : Notepad++

Pour visualiser votre page HTML :

Pour ce qui est du choix du navigateur, il en faut plusieurs ! Chaque site doit être testé au minimum dans Edge, Chrome, Safari et Firefox, ainsi que sur quelques navigateurs mobiles.

Retenez les commandes suivantes :

- Ctrl + R ou F5 : actualiser l'onglet actif.
- Ctrl + F5 : actualiser l'onglet actif et forcer le rafraîchissement du cache.
- Ctrl + + : zoomer la page.
- Ctrl + - : dézoomer la page.
- Ctrl + 0 : revenir au niveau de zoom par défaut.
- Ctrl + U : voir le code source de la page.
- F11 : plein écran.
- F12 : ouvrir les outils de développement.

Pour corriger votre page HTML :

Tous les navigateurs récents intègrent dorénavant des outils de développement, accessibles le plus souvent via le raccourci **F12**. Ces outils sont d'une grande aide à notre travail de développement, car ils permettent d'identifier certaines erreurs de code, d'explorer la structure HTML, de parcourir les règles CSS, de parcourir les erreurs JavaScript, ou encore de vérifier le temps de chargement des fichiers nécessaires à la page.

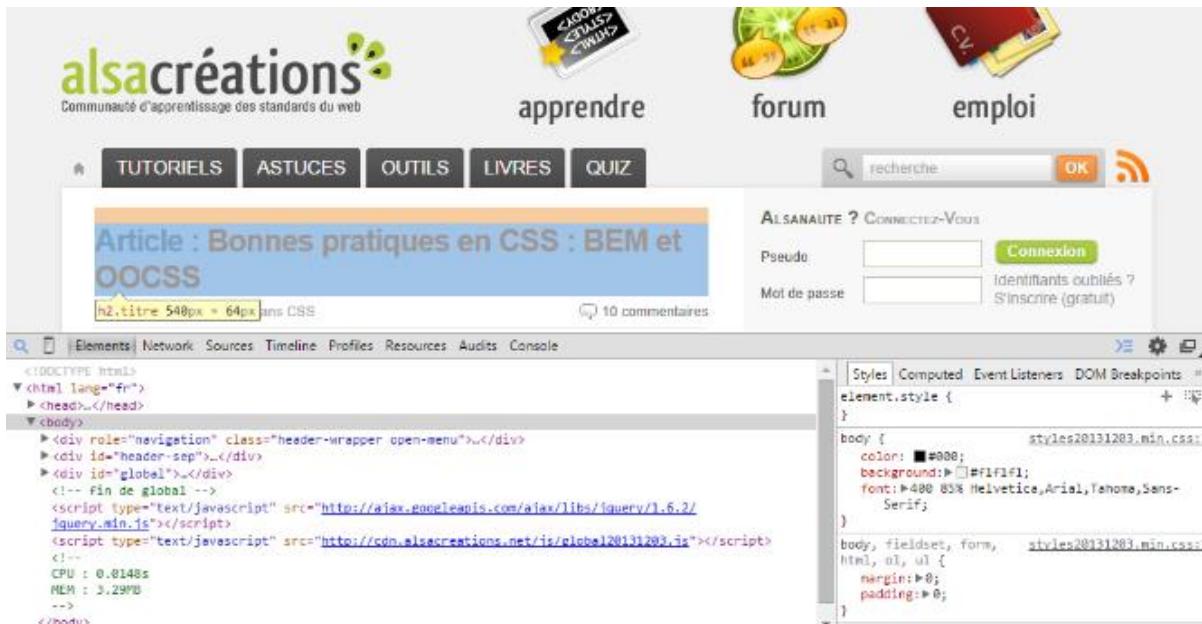


Figure 3 : Le site d'Alsacréation (en haut) passé au crible avec les outils de développement de Google Chrome (en bas)

1.2 Les bases du langage HTML

Les éléments

Grâce à sa syntaxe balisée, HTML est un des langages informatiques les plus faciles à apprendre.

Le code HTML contient les textes de la page web, accompagnés d'**éléments** HTML. Le plus souvent, ces éléments sont composés de 2 **balises** : une ouvrante et une fermante, permettant à ces éléments de s'imbriquer ou de contenir du texte.

Exemple :

```
<div>
    <h1>Titre de la page</h1>
    <p>Premier paragraphe</p>
</div>
```

Les balises HTML sont des instructions réservées au navigateur. Elles se différencient du texte par les caractères < et > appelés **chevrons**.

Le langage HTML propose une centaine d'éléments, presque tous expliqués dans ce syllabus. Chaque élément HTML a une signification : une image , un titre <h1> ou <h2>, un paragraphe <p>, un pied de page <footer>, une citation <cite>, etc.

Retenez que le choix d'un élément se fait toujours en fonction de sa **sémantique** et non pas de son apparence. La sémantique est le sens apporté par un élément. L'élément <cite>, par exemple, indique que son contenu est une citation et provoque la mise en italique de ce contenu. Il ne faut donc surtout pas utiliser <cite> pour mettre du texte en italique si celui-ci n'est pas une citation.

L'**écriture minuscule** est de mise en HTML, bien qu'elle ne soit pas obligatoire en HTML5.

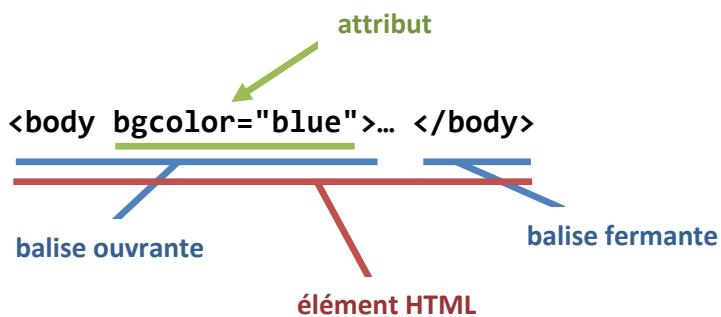
Classification du langage HTML

HTML est donc un **langage balisé** car ses instructions s'écrivent sous forme de balises.

HTML est également un **langage interprété** car il ne nécessite pas de compilation, contrairement à d'autres langages de programmation comme le langage C qui nécessitent un compilateur afin de générer un fichier exécutable (.exe). HTML est directement interprété ligne par ligne, par le navigateur qui joue le rôle d'interpréteur.

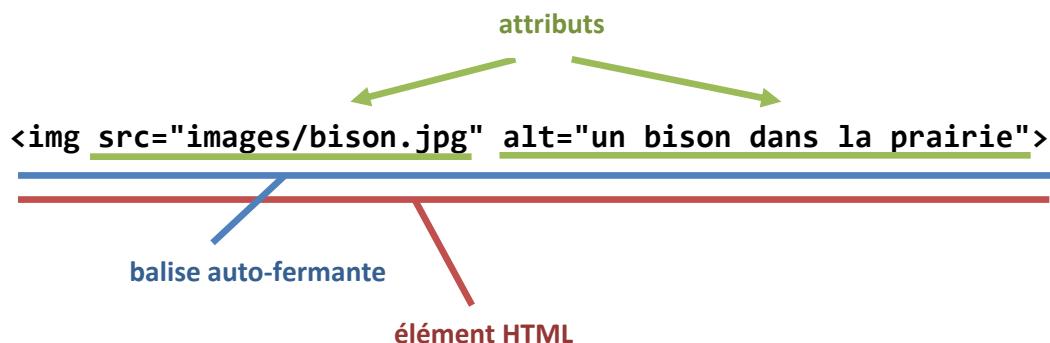
Bien qu'HTML soit un **langage informatique**, on ne peut pas le qualifier de langage de programmation car son rôle se limite à la représentation de données et non pas à l'écriture d'algorithmes.

Les attributs



Dans cet exemple, on distingue ***body*** qui est le nom de l'élément et ***bgcolor*** qui est un attribut de cet élément. Les attributs permettent de personnaliser les éléments en en modifiant certains critères.

Autre exemple, cette fois avec une balise auto-fermante :



Remarques indispensables sur les attributs :

- Chaque élément HTML possède ses propres attributs.
- Chaque attribut possède une valeur par défaut, par exemple "white" pour `bgcolor` qui permet de spécifier la couleur de fond. Il est donc superflu de coder `bgcolor="white"`.
- Les attributs se codent toujours dans la balise ouvrante, jamais dans la balise fermante.
- Il peut y avoir plusieurs attributs codés dans une même balise et ce, dans n'importe quel ordre.
- Un même attribut ne peut être codé qu'une seule fois par élément.
- La valeur de l'attribut doit toujours être entre guillemets.
- Certains attributs sont obligatoires car l'élément n'aurait pas de sens sans (par exemple une image sans sa source `src`) ou car imposé par le W3C (par exemple une image sans texte alternatif `alt`).

Les balises auto-fermantes

Les balises auto-fermantes, appelées aussi balises uniques ou ponctuelles, ne nécessitent pas de fermeture contrairement aux autres balises. Elles n'ont donc pas de contenu. Exemples :

- **
** : retour à la ligne
- **<wbr>** : permission de retour à la ligne si nécessaire, par exemple dans un long mot
- **<hr>** : barre de séparation horizontale
- **** : image
- **<input>** : champ de formulaire
- **<meta>** : informations pour le navigateur et les moteurs de recherche
- **<link>** : inclusion de fichier externe

Attributs de <hr> : size (épaisseur), width (largeur), align (alignement).

Par défaut, le navigateur octroie des valeurs aux attributs non spécifiés. Dans le cas présent, les valeurs par défaut de la balise `<hr>` sont : une épaisseur de trait de 2 pixels, un alignement centré et une largeur de 100%. Seuls les attributs spécifiés dans le code auront une valeur différente de la valeur par défaut.

Il est donc équivalent d'écrire :

- `<hr>`
- `<hr size="2" align="center" width="100%">`

Remarque : en XHTML, l'écriture des balises auto-fermantes nécessite un « / » :

`
 <wbr /> <hr /> <input /> <meta /> <link />`
Cette écriture est permise en HTML5 mais déconseillée.

En revanche, il est erroné de fermer un élément auto-fermant : ``

Les commentaires

Les développeurs utilisent les commentaires pour annuler des parties de code sans les effacer (par exemple lors de la recherche d'un bug), ou encore pour donner des indications facilitant la lisibilité du code (par exemple pour ses collègues ou pour lui-même).

`<!-- Commentaire -->`

La page HTML

Toute page HTML5 contient au minimum ces éléments.

```
<!DOCTYPE html> annonce version html
<html lang="fr">
  <head>
    <meta charset="utf-8"> Format d'encodage
    <title>Code Minimal</title> titre
  </head>
  <body>
    Bonjour !
  </body>
</html>
```

Le résultat affiché ici par votre navigateur sera une page blanche avec le texte « Hello World ! » et ayant pour titre de document « Code Minimal ».

Tout document HTML5 commence par la balise `<!DOCTYPE html>` qui précise aux navigateurs que la version HTML choisie est **HTML5**. Cette balise peut prendre d'autres formes – plus lourdes – pour annoncer des technologies antérieures comme HTML4 ou XHTML :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Nous retrouvons ensuite 3 éléments fondamentaux :

- `<html>`, l'élément racine de la page, qui est unique et qui contient tout le reste du code ;
- `<head>` qui est unique et qui contient toutes les informations non visibles ;
- `<body>` qui est unique et qui contient toutes les informations visibles.

Dans la pratique, les navigateurs récents tolèrent que l'on oublie ces éléments fondamentaux, mais ce serait prendre de mauvaises habitudes que de les oublier.

Nous retrouvons également dans le code ci-dessus l'élément `<title>` qui contient le titre du document, visible dans l'onglet.

Et pour finir, nous retrouvons l'élément `<meta charset="utf-8">` qui précise le format d'encodage. Cet élément aussi a un ancêtre atrocement plus lourd :

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Remarques :

1. L'attribut `lang` dans la balise `<html>` peut être contredit plus loin dans le code si certains textes sont dans une autre langue que celle du document.
2. HTML5 n'est pas sensible à la casse (majuscules/minuscules). Même si en XHTML, le W3C recommande de tout écrire en minuscules, en HTML5 il est équivalent d'écrire `<HTML>`, `<html>`, `<Html>`, etc.
3. **Ne jamais croiser les flux !** `<i>...</i>` est correct alors que `<i>...</i>` est erroné et risque de poser des problèmes.



Traitement des espacements par le navigateur

Le navigateur ne tient compte que d'un seul espace entre les mots. Ainsi :

`mon texte`

s'affiche de façon identique à :

`mon texte`

Le navigateur ne tient pas compte des retours à la ligne dans le code. Pour forcer un retour à la ligne dans la page web, il faut utiliser la baliser `
`. Ainsi :

`mon texte`

s'affiche de façon identique à :

`mon`

`texte`

Le flux

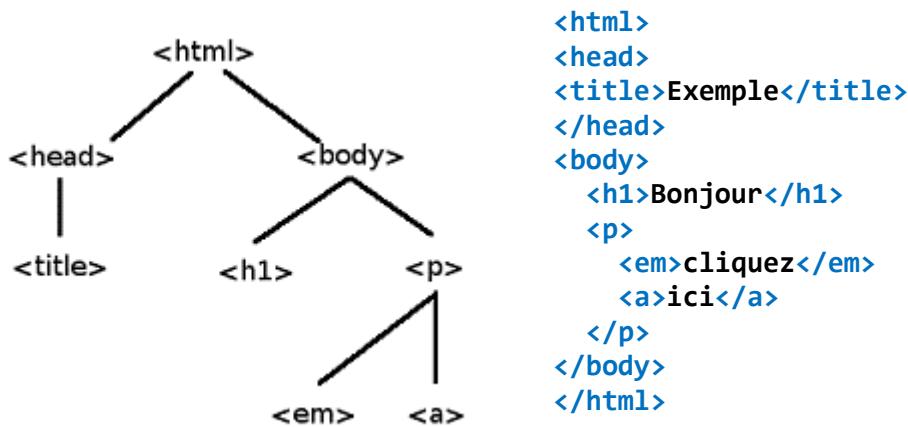
L'ordre dans lequel sont codés les éléments HTML correspond à l'ordre dans lequel ils s'affichent. Cet ordre est appelé le **flux** et suit grossièrement le sens de lecture d'un livre.

L'arborecence

En HTML, l'imbrication des éléments peut être représentée sous forme d'arborescence. Les relations entre les éléments sont alors exprimées avec un vocabulaire proche de celui utilisé en généalogie : les éléments `<body>` et `<head>` sont les enfants de l'élément `<html>`. `<html>` est le parent de `<head>` et `<body>`.

L'élément `<html>` est appelé **l'élément racine** car il est l'ancêtre de tous les autres éléments du document.

Exemple d'arborescence et de code HTML :



Racine : `<html>` est appelé l'élément racine car il contient tous les autres.

Ancêtre : élément qui se trouve sur le chemin entre un élément et l'élément racine.

Dans l'exemple : `<head>` et `<html>` sont les ancêtres de `<title>`.

Parent : ancêtre direct d'un élément.

Dans l'exemple : `<head>` est le parent de `<title>`.

Enfant : inverse d'un parent. Descendant direct d'un élément.

Dans l'exemple : `<h1>` et `<p>` sont les enfants de `<body>`.

Descendant : inverse de l'ancêtre.

Dans l'exemple : `<h1>`, `<p>`, `` et `<a>` sont les descendants de `<body>`.

Frère : deux enfants ayant le même parent sont des frères.

Dans l'exemple : `<h1>` et `<p>` sont frères.

1.3 Les formats d'encodage

Pour enregistrer un texte ou une suite de caractères en mémoire, il faut passer par une table de correspondance entre caractères et valeurs numériques. Ces tables sont des formats d'encodage.

Bien avant l'HTML, dans les années 1960, est apparu l'ASCII, la première véritable norme de codage de caractères. L'ASCII est une table de correspondance dans laquelle chaque caractère correspond à une valeur numérique : le caractère « A » vaut 65 ou 1000001 en binaire, valeur qui est enregistrée en mémoire pour représenter la lettre « A ».

Les fichiers HTML ne dérogent pas aux formats d'encodage : leur lecture et leur écriture se fait selon un format d'encodage à définir :

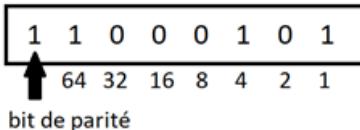
- Lors de l'édition du code HTML, le format d'encodage est paramétrable dans l'éditeur de code.
- Lors de l'interprétation de la page, le navigateur se fie à la mét-balise `charset`.

De l'ASCII à l'UTF-8

ASCII : *American Standard Code for Information Interchange* (Code standard américain pour l'échange d'informations). L'ASCII est aussi appelé « code ISO à 7 bits », l'ISO étant l'Organisation Internationale pour la Normalisation (*International Standards Organization*).

L'ASCII est une norme américaine définissant le codage des caractères non accentués de l'alphabet latin, des chiffres et d'un certain nombre de signes usuels en informatique. Il s'agit de la norme de codage de caractères en informatique la plus connue et la plus compatible.

L'ASCII est codé sur 7 bits (128 caractères), complétés par un huitième bit, redondant, pour effectuer un contrôle de parité : grâce à ce système de codage, tout caractère comportant un nombre impair de 1 est détecté et une erreur est signalée.

En ASCII, la lettre "E" correspond au code 69							
Voici son enregistrement en mémoire sur 1 octet :							
							
1	1	0	0	0	1	0	1
64	32	16	8	4	2	1	
bit de parité							

ASCII étendu ANSI : Comme le code ASCII, mis au point pour la langue anglaise, ne contient pas de caractères accentués, plusieurs extensions de l'ASCII ont vu le jour. Ces extensions codées sur 8 bits au lieu de 7 permettent de coder plus de caractères. Cependant, les extensions du code ASCII sont nombreuses et varient selon les pays. Ce qui n'est pas pratique pour les échanges de documents ! C'est pourquoi il existe un format d'encodage plus universel : l'UNICODE.

Unicode : L'Unicode (1991) est un système de codage des caractères sur plusieurs octets.

Ce format d'encodage regroupe la quasi-totalité des alphabets de la planète (arabe, arménien, cyrillique, grec, hébreu, latin, japonais, syriaque, thâna, n'ko, samaritain, mandéen, bengali, gourmoukhî, oriya, tamoul, thaï, lao, tibétain, éthiopien, birman, kmer, mongol, runique, etc.) mais aussi le braille, le phonétique, les symboles monétaires, des symboles techniques divers, des formes géométriques, des flèches, les pièces du jeu d'échec, des petits bonhommes, les signes du zodiaque, des symboles religieux et astrologiques, etc.

Même si l'UNICODE est très complet, il reste peu utilisé par rapport à l'ASCII à cause de son problème de surpoids : en Unicode chaque caractère prend 2, 3, 4 octets ou plus. Autrement dit, le moindre texte prend deux fois plus de place qu'en ASCII. C'est du gaspillage.

UTF-8 : L'UTF-8 est la solution au problème de poids de l'UNICODE et aux limites de l'ASCII.

En UTF-8, les caractères ASCII sont codés sur 1 octet ; les caractères plus rares signalés par le 8^e bit de l'ASCII (celui qui servait au contrôle de parité) codés en UNICODE (2 octets ou +).

Par exemple, pour le texte "Bienvenue chez Sébastien !", seul le "é" ne fait pas partie du code ASCII et est codé sur 2 octets.

L'UTF-8 rassemble donc le meilleur des deux mondes : l'efficacité de l'ASCII et l'étendue de l'Unicode. C'est pourquoi l'UTF-8 a été adopté comme norme pour XML et HTML5.



Définition du nombre d'octets utilisés	
Représentation binaire UTF-8	Signification
0xxxxxxx	1 octet codant 1 à 7 bits
110xxxxx 10xxxxxx	2 octets codant 8 à 11 bits
1110xxxx 10xxxxxx 10xxxxxx	3 octets codant 12 à 16 bits
11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	4 octets codant 17 à 21 bits

UTF-16 et UTF-32 : les formats d'encodage UTF-16 et UTF-32 ne sont pas des mises à jour de l'UTF-8, mais des versions avec un minimum de 16 ou 32 octets par caractères, donc plus lourdes, donc inadaptées au Web.

En conclusion, dans nos pages web comment allons-nous faire ?

Nous travaillerons en HTML5 et choisirons le format d'encodage UTF-8. Avant de coder chaque page, nous vérifierons 2 choses :

1. Le format d'encodage des caractères de l'éditeur : UTF-8 (sans BOM de préférence)
2. La présence d'une méta-balise annonçant le format d'encodage UTF-8 dans le *head*.

```

Fichier Edition Recherche Affichage Format Langage Paramétrage Macro Exécution TextFX Plugins Document ?
Convertir en Format Windows
Convertir en Format UNIX
Convertir en Format Mac
Encoder en ANSI
Encoder en UTF-8 (sans BOM)
Encoder en UTF-8
Encoder en UCS-2 Big Endian
Encoder en UCS-2 Little Endian
Convertir en ANSI
Convertir en UTF-8 (sans BOM)
Convertir en UTF-8
Convertir en UCS-2 Big Endian
Convertir en UCS-2 Little Endian
r : portfolio</h1>
.php">accueil </a><a href="curriculu
or die ("Echec de connexion a MySQL
Echec de connexion a la BD <br>");
$result_admin=mysql_query("select * from admin");

```

Exemple de paramétrage du format d'encodage dans Notepad++

Les entités de caractères

Les premières versions du langage HTML travaillaient en ASCII (comme le langage C) et nous privaient donc de certains caractères spéciaux ou accentués comme le « é » pour lesquels il fallait passer par des codes particuliers appelés **entités de caractères**. Ces entités constituent une solution pour coder des caractères spéciaux indépendamment du format d'encodage.

Les entités de caractères datent du SGML, ancêtre de HTML, et restent utilisées dans certains cas, mais heureusement pas pour coder chaque caractère accentué.

Exemples : **&nbsp** ou son code numérique ** ** représente un espace insécable.

€ ou **€** représente le caractère €.

&amp ou **&** représente le caractère &.

1.4 Les familles d'éléments

Les éléments HTML se classent en 2 grandes familles : les éléments **lignes** et les éléments **blocs**.
Une règle d'or en HTML :

Les lignes ne peuvent pas contenir de blocs !

Les éléments lignes

Les éléments lignes (ou en-lignes) ont pour but de donner plus de sens à certaines parties d'un texte, voir d'en modifier l'apparence.

La taille des éléments lignes est déterminée par leur contenu. Les éléments lignes se disposent côté à côté, de gauche à droite, en ne provoquant un retour à la ligne que si l'espace est insuffisant.



Les principaux éléments lignes sont : ``, ``, `<label>`, `<input>`, `<select>`, `<textarea>`, `<i>`, ``, ``, `<mark>`, `<cite>`, `<abbr>`, ``, `<video>`, `<audio>`, `<svg>`, `<canvas>`, `<iframe>`.

Trois éléments lignes font exception à la règle car ils peuvent contenir des blocs depuis l'HTML5 : `<a>`, `<ins>` et `` qui sont autorisées à contenir des blocs depuis HTML5.

Les éléments blocs

Les éléments blocs ont pour but de structurer la page en zones rectangulaires. Ils s'affichent les uns sous les autres, en s'étendant sur toute la largeur, avec une hauteur dépendant de leur contenu.

Par défaut, de nombreux éléments blocs possèdent des marges internes et externes non nulles. Ce détail est important car ces marges risquent de vous surprendre.



Les principaux éléments blocs sont : `<div>`, `<main>`, `<header>`, `<footer>`, `<article>`, `<section>`, `<aside>`, `<dialog>`, `<form>`, `<pre>`, `<address>`, ``, ``, ``, `<dl>`, `<dt>` et `<dd>`.

Certains éléments blocs font exceptions à la règle car ils ne peuvent pas contenir d'autres blocs : `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>` et `<p>`.

Nouvelles catégories HTML5



Cette séparation entre lignes et blocs est en réalité une version simplifiée des catégories d'éléments. HTML5 bouleverse cette répartition : les **blocs** et **lignes** deviennent les **flux** et **phrases** et entretiennent à peu de choses près les mêmes rôles.

Certains éléments comme `<a>`, `<ins>` et `` font partie dorénavant des 2 catégories.

Parmi les **flux**, on retrouve 2 sous-catégories :

- Les **titres** (*heading content*) comme `<h1>`, `<h2>`, etc.
- Les **sectionnements** (*sectioning content*) comme `<article>`, `<section>`, `<nav>` et `<aside>`.

Parmi les autres familles, on retrouve :

- Les **phrases**
- Les **embarqués** (*embedded content*) : éléments importants du contenu dans le document, comme ``, `<video>`, `<object>`, `<audio>`, `<embed>`, `<iframe>`, `<svg>` et `<math>`.
- Les **interactifs** (*interactive content*) comme `<a>`, `<audio>` si l'attribut `controls` est précisé, `<button>`, `<details>`, `<embed>`, `<iframe>`, `` si l'attribut `usemap` est précisé, `<input>` si l'attribut `type` ne vaut pas `hidden`, `<keygen>`, `<label>`, `<menu>`, `<object>` si l'attribut `usemap` est précisé, `<select>`, `<textarea>`, `<video>` si l'attribut `controls` est précisé.
- Les **métadonnées** (*metadata content*).

1.5 Les principaux éléments HTML

Les titres et les paragraphes

Les titres se codent à l'aide des éléments **<h1>** à **<h6>**.

Les paragraphes de texte se codent avec **<p>**.

L'écriture de votre page web doit s'accompagner d'une démarche d'identification du titre principal **<h1>**, des sous-titres **<h2>**, des sous-sous-titres **<h3>**, des sous-sous-sous-titres **<h4>**, etc.

<h1>Pat Roach</h1>

h1

<h2>Biographie</h2>

h2

<p>Pat Roach est un acteur et catcheur britannique (1937-2004).</p>

<h2>Au cinéma</h2>

<p>Connu pour ses rôles de méchants célèbres, Pat affronte plusieurs fois Harrison Ford dans la saga *<i>Indiana Jones</i>*, Sean Connery dans le James Bond *<i>Jamais plus jamais</i>* et Val Kilmer dans *<i>Willow</i>*.</p>

Par page, le titre **<h1>** est sensé être unique, exception faite de certaines structures de pages décomposées en sections.

Un titre **<h4>** a, par défaut, la même taille que le texte. Les titres de niveau 5 et 6 étant plus petits que le texte de la page, ils sont considérés comme moins importants que le texte par les moteurs de recherche. Rappelons que le choix de l'élément ne se fait pas en fonction de son rendu, mais en fonction de sa sémantique.

Ces 7 éléments présentent une particularité : ce sont des blocs, mais ils ne peuvent pas contenir d'autres blocs. Ils peuvent cependant contenir des éléments lignes comme **<i>** ou ****.

L'élément **<p>** présente une autre particularité : on peut omettre de le fermer. Le navigateur considère l'élément fermé dès qu'il rencontre la fermeture de son parent ou qu'il rencontre le début d'un nouveau **<p>**. Cette particularité se retrouve dans les éléments : **<p>**, **<dd>**, **<dt>**, ****, **<optgroup>**, **<option>**, **<rt>**, **<rp>**, **<td>**, **<th>**, **<tr>**, **<thead>** et **<tfoot>**.

Les éléments d'enrichissement du texte

...	Ligne neutre : élément sans sémantique, destiné à être mis en page avec du CSS.
...	Texte renforcé : applique du gras et augmente l'importance de son contenu (référencement).
<q>...</q>	Citation courte : citation au sein d'un texte (entraîne l'apparition de guillemets). Pour un bloc de citation, utilisez <blockquote>.
<cite>...</cite>	Titre d'œuvre : pour citer une œuvre (livre, chanson, film, peinture, site web, jeu, article de recherche, etc.)
<code>...</code>	Extrait de code source
<kbd>...</kbd>	Commandes Utilisateur : applique une apparence de touche de clavier.
<samp>...</samp>	Échantillon produit : un résultat produit par un programme informatique
<var>...</var>	Variable : variable mathématique ou de programmation
<abbr title="">...</abbr>	Abbréviation : Définit une abréviation et laisse apparaître des informations supplémentaires (<i>title</i>) au survol.
<small>...</small>	Petit texte
<sub>...</sub>	Texte en indice
<sup>...</sup>	Texte en exposant
...	Texte en gras (<i>bold</i>)
<i>...</i>	Texte <i>italique</i>
...	Texte <i>emphasé</i>
<mark>...</mark>	Texte surligné : Texte surligné
<s>...</s>	Texte barré : (<i>striked</i>) pour indiquer un contenu qui n'est plus d'actualité
<u>...</u>	Texte souligné : (<i>underlined</i>) pour signaler une faute dans un texte, surtout pas pour des titres !
<ins>...</ins> ...	Texte inséré ou supprimé : utilisé pour corriger un texte en copie.

Les éléments de structure de la page

Bloc neutre	<code><div>...</div></code>	Aucune sémantique.
Bloc principal	<code><main>...</main></code>	Bloc principal
Bloc d'en-tête	<code><header>...</header></code>	En-tête de la page ou d'un article
Bloc de pied de page	<code><footer>...</footer></code>	Pied de la page ou d'un article
Bloc de section	<code><section>...</section></code>	Section de la page ou d'un article. Nécessite un titre.
Bloc d'article	<code><article>...</article></code>	Article, afin de séparer différents sujets d'une page. Nécessite un titre.
Bloc d'aparté	<code><aside>...</aside></code>	Contenu entretenant un lien indirect avec le sujet et dont la lecture n'est pas essentielle à la compréhension du sujet.
Bloc de dialogue (<i>chat</i>)	<code><dialog>...</dialog></code>	
Bloc de navigation	<code><nav>...</nav></code>	
Adresse	<code><address>...</address></code>	

Les éléments interdits

Ces éléments permettent de mettre en forme un texte ou une partie de texte. **Nous ne les utiliserons pas !** Pourquoi ? Parce que la mise en page est l'affaire du CSS !

Centré	<code><center>...</center></code>	Elément centré
Barré	<code><strike>...</strike></code>	Texte barré
Mono espacé	<code><tt>...</tt></code>	Texte mono espacé
Grand	<code><big>...</big></code>	Grand texte
Acronyme	<code><acronym>...</acronym></code>	Définition d'accronyme. Utiliser <code><abbr></code> à la place.
Texte clignotant	<code><blink>...</blink></code>	Texte clignotant.
Texte défilant	<code><marquee>...</marquee></code>	Texte défilant.
Taille et couleur de la police	<code>...</code>	La balise <code></code> n'a pas d'action spécifique. Ce sont ses attributs (color, size, etc.) qui permettent de modifier l'apparence du texte.
Code ASCII	<code><xmp>...</xmp></code> <code><listing>...</listing></code>	Équivalent à <code><pre>...</pre></code> mais déconseillé. <code><xmp></code> et <code><listing></code> ont pour différence le nombre de caractères par ligne (80 pour XMP et 132 pour LISTING).

Quelques éléments spéciaux

<pre>...</pre> permet d'afficher le texte dans son état brut, sans tenir compte des restrictions du code HTML (plus de problèmes avec les caractères spéciaux, etc.)

<script type="text/javascript"></script>

Allégé de son attribut *type* désormais obsolète, l'élément `<script>` sert à insérer du JavaScript ou à lier un fichier JavaScript externe.

<link rel="stylesheet" href="..." type="text/css">

Allégé de son attribut *type* désormais obsolète, l'élément `<link>` sert surtout à lier un fichier CSS externe à la page HTML, mais peut aussi lier d'autres ressources.

<object></object> et **<embed></embed>** permettent d'insérer des ressources externes.

<style></style> permet d'insérer du CSS directement dans la page.

<time datetime=""></time> définit une date, une heure, ou les deux. La même date peut être codée dans l'attribut *datetime* avec une syntaxe permettant aux moteurs de recherche de la comprendre.

<canvas></canvas> apporte la possibilité de dessiner directement dans le navigateur, ouvrant ainsi la voie à des animations graphiques complexes pilotées en JavaScript.

<progress> permet de coder une barre d'avancement de tâche.

`<progress id="progressBar" value="66.6" max="100"></progress>`



<meter> permet de visualiser une mesure sous forme d'un thermomètre horizontal. Les attributs *low*, *high* et *optimum* servent à modifier la couleur du thermomètre (rouge, jaune, vert). Le contenu de `meter` peut être écrit de bien des manières : valeur numérique, fraction, pourcentage, etc.

`<meter min="0" max="10" value="8" low="5" high="6" optimum="8">8/10</meter>`



<ruby>, <rt> (*ruby text*) et **<rp>** (*ruby parentheses*) permettent des annotations ruby employées en Asie de l'Est pour donner la prononciation de caractères asiatiques.

Ceci est une

`<ruby>annotation<rp>(</rp><rt>[phonétique]</rt><rp>)</rp></ruby>` ruby.

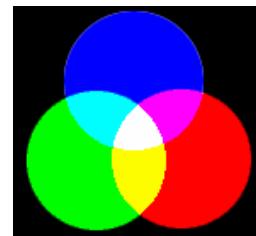
<details> et **<summary>** permettent de baliser un complément d'informations. Ce complément d'informations peut être déployé (visible) ou compacté (invisible). Une petite flèche ► indique l'état. Cet état peut changer à la suite d'un clic. La présence de l'attribut *open* indique que le contenu est déployé, son absence indique que le contenu est compacté.

```
<details open>
    <summary>Cliquer ici pour déployer</summary>
    Informations à déployer (peut contenir du code HTML)
</details>
```

1.6 Les couleurs

En HTML, il existe deux syntaxes pour coder les couleurs : par certains noms anglais (liste disponible plus loin) ou par leurs codes hexadécimaux.

Coder des couleurs par leurs noms



Les noms de couleurs en anglais sont plus explicites que les codes hexadécimaux, mais ils sont parfois incompatibles et sont surtout beaucoup moins nombreux : seulement une centaine contre les 16 millions de couleurs possibles en hexadécimal.

Exemples : *antiquewhite, aqua, aquamarine, azure, beige, black, blue, brown, green, grey (ou gray), fuchsia, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow, etc.*

Coder des couleurs par code hexadécimal

C'est le système de la synthèse additive RVB : partant du noir (absence de couleur), des composantes de couleur primaire sont ajoutées pour tendre vers le blanc. Les couleurs primaires sont dans ce cas le rouge, le vert et le bleu.

Le code hexadécimal commence par un croisillon « # » et est composé de 6 chiffres allant de 0 à ... F ! Les 2 premiers chiffres représentent la composante rouge, les 2 chiffres du milieu la composante verte et les 2 chiffres de la fin la composante bleue.

En hexadécimal on compte 0 1 2 3 4 5 6 7 8 9 A B C D E F... F valant 15. Après F, on reporte au rang suivant : 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F. 1F en hexadécimal vaut alors $1 \times 16 + 15$, c'est-à-dire 31 en décimal.

Chaque composante a une valeur comprise entre 00 à FF, couvrant ainsi 256 possibilités (FF = $16 \times 15 + 15 = 255$). Le nombre de couleurs possibles est de $256 \times 256 \times 256 = 16\,777\,216$.

Exemples

Bleu	#0000FF	Blanc	#FFFFFF
Rouge	#FF0000	Gris clair	#C0C0C0
Vert	#00FF00	Violet	#8000FF
Jaune	#FFFF00	Noir	#000000

Une fois familier avec l'écriture hexadécimale, il est facile d'ajuster une couleur. Par exemple pour assombrir du rouge, le code #FF0000 pourrait devenir #BB0000 ou #880000.

Alors que pour l'éclaircir, comme FF est la valeur maximale d'une composante, nous pourrions augmenter les 2 autres composantes de manière égale : #FF4444.

Si nous souhaitions ternir le rouge (le rendre grisé), il suffit de rapprocher les valeurs des 3 composantes : #CC3333. Ce qui donne un rouge moins éclatant que le rouge vif #FF0000.

Les couleurs ayant les 3 composantes égales sont grises, puisqu'aucune composante ne l'emporte sur les autres.

Noms de couleurs



Nom Aperçu	RVB	Nom Aperçu	RVB
aliceblue	#F0F8FF	lightsalmon	#FFA07A
antiquewhite	#FAEBD7	lightseagreen	#20B2AA
aqua	#00FFFF	lightskyblue	#87CEFA
aquamarine	#7FFFDD	lightslategray	#778899
azure	#F0FFFF	lightsteelblue	#B0C4DE
beige	#F5F5DC	lightyellow	#FFFFE0
bisque	#FFE4C4	lime	#00FF00
black	#000000	limegreen	#32CD32
blanchedalmond	#FFEBCD	linen	#FAF0E6
blue	#0000FF	magenta	#FF00FF
blueviolet	#8A2BE2	maroon	#800000
brown	#A52A2A	mediumaquamarine	#66CDAA
burlywood	#DEB887	mediumblue	#0000CD
cadetblue	#5F9EA0	mediumorchid	#BA55D3
chartreuse	#7FFF00	mediumpurple	#9370DB
chocolate	#D2691E	mediumseagreen	#3CB371
coral	#FF7F50	mediumslateblue	#7B68EE
cornflowerblue	#6495ED	mediumspringgreen	#00FA9A
cornsilk	#FFF8DC	mediumturquoise	#48D1CC
crimson	#DC143C	mediumvioletred	#C71585
cyan	#00FFFF	midnightblue	#191970
darkblue	#00008B	mintcream	#F5FFFA
darkcyan	#008B8B	mistyrose	#FFE4E1
darkgoldenrod	#B8860B	moccasin	#FFE4B5
darkgray	#A9A9A9	navajowhite	#FFDEAD
darkgreen	#006400	navy	#000080
darkkhaki	#BDB76B	oldlace	#FDF5E6
darkmagenta	#8B008B	olive	#808000
darkolivegreen	#556B2F	olivedrab	#6B8E23
darkorange	#FF8C00	orange	#FFA500
darkorchid	#9932CC	orangered	#FF4500
darkred	#8B0000	orchid	#DA70D6
darksalmon	#E9967A	palegoldenrod	#EEE8AA
darkseagreen	#8FBBC8	palegreen	#98FB98
darkslateblue	#483D8B	paleturquoise	#AFEEEE
darkslategray	#2F4F4F	palevioletred	#DB7093
darkturquoise	#00CED1	papayawhip	#FFEFBD
darkviolet	#9400D3	peachpuff	#FFDAB9
deeppink	#FF1493	peru	#CD853F
deepskyblue	#00BFFF	pink	#FFC0CB
dimgray	#696969	plum	#DDA0DD
dodgerblue	#1E90FF	powderblue	#B0E0E6
firebrick	#B22222	purple	#800080
floralwhite	#FFFFAF	red	#FF0000

forestgreen		#228B22	rosybrown		#BC8F8F
fuchsia		#FF00FF	royalblue		#4169E1
gainsboro		#DCDCDC	saddlebrown		#8B4513
ghostwhite		#F8F8FF	salmon		#FA8072
gold		#FFD700	sandybrown		#F4A460
goldenrod		#DAA520	seagreen		#2E8B57
gray		#808080	seashell		#FFF5EE
green		#008000	sienna		#A0522D
greenyellow		#ADFF2F	silver		#C0C0C0
honeydew		#F0FFF0	skyblue		#87CEEB
hotpink		#FF69B4	slateblue		#6A5ACD
indianred		#CD5C5C	slategray		#708090
indigo		#4B0082	snow		#FFFFFA
ivory		#FFFFFF0	springgreen		#00FF7F
khaki		#F0E68C	steelblue		#4682B4
lavender		#E6E6FA	Tan		#D2B48C
lavenderblush		#FFF0F5	teal		#008080
lawngreen		#7CFC00	thistle		#D8BFD8
lemonchiffon		#FFFACD	tomato		#FF6347
lightblue		#ADD8E6	turquoise		#40E0D0
lightcoral		#F08080	violet		#EE82EE
lightcyan		#E0FFFF	wheat		#F5DEB3
lightgoldenrodyellow		#FAFAD2	white		#FFFFFF
lightgreen		#90EE90	whitesmoke		#F5F5F5
lightgrey		#D3D3D3	yellow		#FFFF00
lightpink		#FFB6C1	yellowgreen		#9ACD32

Le site W3Schools propose un excellent Color Picker en ligne :

https://www.w3schools.com/colors/colors_picker.asp.

Adobe Color est un outil sympa pour aider à choisir plusieurs couleurs en harmonie :

<https://color.adobe.com/fr/create/color-wheel/>.

1.7 Les listes

Listes ordonnées et non-ordonnées

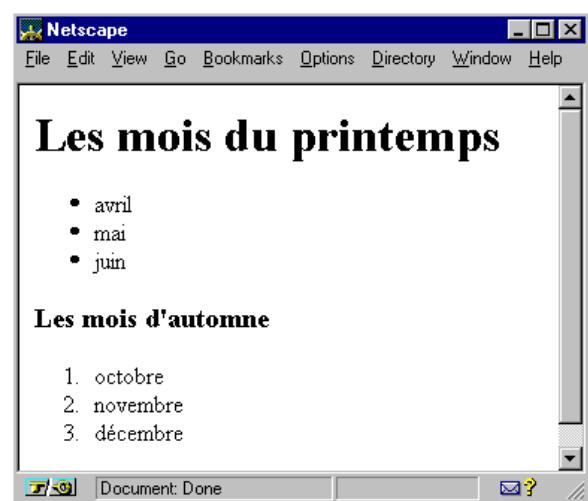
Liste ordonnée `...` (*Ordered List*) Liste dont les éléments sont numérotés.

Liste non-ordonnée `...` (*Unordered List*) Liste dont les éléments sont pucés (disque, cercle ou carré).

Élément de liste `...` (*List Item*) Chaque élément de la liste.

Exemple :

```
<h1>Les mois du printemps</h1>
<ul>
  <li>avril</li>
  <li>mai</li>
  <li>juin</li>
</ul>
<h3>Les mois d'automne</h3>
<ol>
  <li>octobre</li>
  <li>novembre</li>
  <li>décembre</li>
</ol>
```



Remarques :

- La dénomination « ordonné », ne signifie pas que la liste a été triée, mais que les éléments de la liste sont numérotés.
- Les listes peuvent s'imbriquer :

```
<h4>Les 12 mois</h4>
<ul>
  <li>Les mois du printemps
    <ol>
      <li>avril</li>
    </ol>
  </li>
</ul>
```

Les 12 mois

- Les mois du printemps
 - 1. avril

Attributs des listes ordonnées `` :

- **type** : apparence des puces : A (pour des lettres majuscules), a (lettres minuscules), I (nombres romains majuscules), i (nombres romains minuscules) ou 1 (nombres). Par défaut, **type** vaut 1.
- **start** : valeur de départ de la numérotation.
- **reversed** : inverse la numérotation (attribut à utiliser sans valeur).

Attribut des listes non-ordonnées ** :

- **type** : apparence des puces : *disc*, *circle* ou *square*.

Attributs des éléments de listes ** :

- **value** : définit le numéro d'un élément de liste (par exemple pour sauter de l'élément 5 à l'élément 8).

Exemple :

```
<h2>Liste quelconque</h2>
<ol type="a" start="2">
  <li>Truc</li>
  <li>Chose</li>
  <li value="8">Bidule</li>
</ol>
```

Liste quelconque

- b. Truc
- c. Chose
- h. Bidule

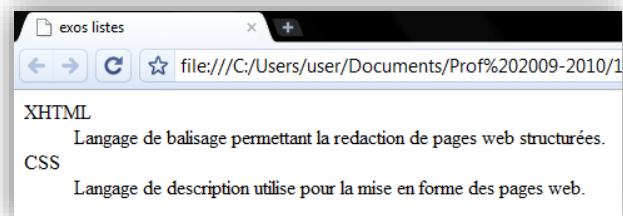
Listes de définitions ou d'associations

L'élément *<dl>* (*definition list*) permet de coder une liste d'une ou plusieurs définitions. Chaque mot à définir est contenu dans un élément *<dt>* (*definition term*) et chaque définition par dans un élément *<dd>* (*definition definition*).

En HTML5, ces éléments ont vu leur sens s'élargir à n'importe quelle liste d'associations. Plutôt que des termes et leurs définitions, on peut dorénavant y stocker des livres et leurs auteurs, des films et leurs réalisateurs, des cours et leurs professeurs, etc.

Exemple :

```
<dl>
  <dt>XHTML</dt>
  <dd>Langage de balisage permettant la redaction de pages web structurées.</dd>
  <dt>CSS</dt>
  <dd>Langage de description utilise pour la mise en forme des pages web.</dd>
</dl>
```



Exercice : Menu du jour

A l'aide des balises ``, ``, ``, ainsi que des balises `` (gras), `<i>` (italique) et `<h1>` (titre), créez une page HTML proposant le même menu du jour et la même mise en page que dans l'encadré ci-contre.

Assurez vous d'avoir respecté toutes les règles de syntaxe HTML vues, ainsi que les règles de parenté des éléments ``, `` et `` !

Menu du jour

- **Potage** : velouté de chicons
- **Plat du jour :**
 - i. Boulettes sauce tomate, frites
 - ii. Pied de porc aux échalottes
 - iii. Lasagne
- **Dessert** : *Baklava*

1.8 Les tableaux

L'usage des tableaux en HTML sert à aligner des données selon un repaire à 2 dimensions (lignes / colonnes). Les tableaux HTML ne doivent surtout pas être utilisés pour créer la structure d'une page web, mais seulement pour structurer des données au sein d'une page.

En HTML, comme les éléments ne peuvent pas se croiser, nous ne parlerons pas de lignes et de colonnes, mais de lignes et de cellules : les tableaux sont divisés en lignes et chaque ligne est divisée en cellules.

Tableau	<code><table>...</table></code>	Début et fin d'un tableau
Ligne	<code><tr>...</tr></code>	(table row) Début et fin d'une ligne
Cellule	<code><td>...</td></code>	(table data) Début et fin d'une cellule
Cellule d'en-tête	<code><th>...</th></code>	(table header) Idem que <td> avec le texte en gras. Permet de différencier une cellule plus importante que les autres.
Légende	<code><caption>...</caption></code> Permet d'afficher un commentaire, une légende.	

Remarque : les tableaux et les cellules supportent bon nombre d'attributs pour en paramétrier l'affichage, mais tous sont dépréciés - à part *colspan* et *rowspan* – en faveur de propriétés CSS.

Exemple 1 : tableau de 2 lignes et 6 cellules

```
<table border="1" cellspacing="0" cellpadding="4">
  <tr>
    <td>Cellule 1</td>
    <td>Cellule 2</td>
    <td>Cellule 3</td>
  </tr>
  <tr>
    <td>Cellule 4</td>
    <td>Cellule 5</td>
    <td>Cellule 6</td>
  </tr>
</table>
```

Cellule 1	Cellule 2	Cellule 3
Cellule 4	Cellule 5	Cellule 6

Remarque : les cellules d'un tableau peuvent contenir n'importe quel élément HTML : du texte, des images, des liens et même d'autres tableaux.

Fusion de cellules

Les cellules de tableau peuvent fusionner. Une fusion horizontale s'obtient avec ***colspan***. Une fusion verticale s'obtient avec ***rowspan***.

Pour fusionner des cellules, conservez la première de ces cellules, codez-y l'attribut ***colspan*** ou ***rowspan*** avec pour valeur le nombre de cellules fusionnant. Puis, supprimez les autres cellules à fusionner.

Exemple 2 : fusion horizontale

```
<table border="1" cellspacing="0" cellpadding="4">
<tr>
    <td colspan="2">Cellule 1</td>
    <td>Cellule 2</td>
    <td>Cellule 3</td>
</tr>
<tr>
    <td>Cellule 4</td>
    <td>Cellule 5</td>
    <td>Cellule 6</td>
</tr>
</table>
```

Cellule 1	Cellule 3	
Cellule 4	Cellule 5	Cellule 6

Exemple 3 : fusion verticale

```
<table border="1" cellspacing="0" cellpadding="4">
<tr>
    <td rowspan="2">Cellule 1</td>
    <td>Cellule 2</td>
    <td>Cellule 3</td>
</tr>
<tr>
    <td>Cellule 4</td>
    <td>Cellule 5</td>
    <td>Cellule 6</td>
</tr>
</table>
```

Cellule 1	Cellule 2	Cellule 3
	Cellule 5	Cellule 6

Exercice : Pizza ou choucroute

Menu			
Semaine	Plats		Desserts
	■ Pizza	■ Salade	○ Tarte
Weekend	Plats		Desserts
	■ Couscous	■ Salade	○ Glace
		■ Lasagne	○ Mousse au chocolat

Exercice : Animaux de la forêt

Animaux de la forêt		
Grosses bêtes		Petites bêtes
a. élans b. ours c. chevreuils d. sangliers	a. renards b. lynx	a. lièvres b. hérissons c. écureuils d. blaireaux

Légende : animaux du bois de Sherwood

1.9 Les liens

L'hypertexte

L'hypertexte est un concept d'agencement des informations de façon à les lier entre elles par des hyperliens, en opposition avec la linéarité des livres, des discours ou des vidéos.

HTML (*Hyper Text Markup Language*) est un langage hypertexte (et hypergraphique) qui permet de cliquer sur un mot ou une image afin d'accéder à de nouvelles informations.



Ce concept d'hyperliens a été imaginé en 1945 par un ingénieur américain du nom de Vannevar Bush qui avait imaginé à l'époque la façon dont devrait s'agencer les informations dans un système servant d'extension à la mémoire humaine et permettant de stocker de grandes quantités d'informations. Pour ce système, l'ingénieur s'inspira du fonctionnement du cerveau humain, et bien que les moyens techniques de l'époque ne lui permettent pas de le réaliser, il coucha son idée sur le papier, devenant ainsi un des principaux précurseurs du Web.

Les liens

La balise HTML permettant la création de ces liens hypertextes est la balise `<a>`.

Exemples :

```
<a href="index.html" title="Retour à l'accueil">Accueil</a>
<a href="doc/fichier.pdf" target="_blank" rel="noopener">Règlement</a>
```

Attributs de l'élément `<a>` :

- **`href`** (pour *Hypertext REference*) contient l'adresse (URL) du document ciblé.
- **`title`** (facultatif) : contient une explication sur la cible du lien. Cette explication apparaîtra lors du survol prolongé du lien. Il est vivement déconseillé de coder un attribut `title` identique à l'intitulé du lien.
- **`target`** (facultatif) précise la cible du lien et peut prendre les valeurs :
 - `"_blank"` qui indique au navigateur qu'il doit créer une nouvelle fenêtre afin d'y afficher le fichier. Cette valeur doit faire intervenir un attribut `rel="noopener"` pour raison de sécurité.
 - `"_self"` qui indique que le fichier sera chargé dans la même fenêtre que celle dans laquelle se trouve le lien.
 - `"_top"` qui implique l'affichage du fichier sur toute la surface de la fenêtre du browser.
 - Dans le cas des *frames*, l'attribut `target` peut également prendre comme valeur le nom d'un *frame*.

Envoi d'email via un lien



Il est possible d'envoyer un email grâce à un lien :

```
<a href="mailto:adresse"> envoyer un email </a>
```

Les protocoles HTTP et HTTPS

Lorsque vous chargez une page web, le navigateur envoie une requête sur le réseau, en suivant scrupuleusement un protocole appelé **HTTP** (*HyperText Transfer Protocol*) ou **HTTPS** pour sa version plus sécurisée (S pour *Secure*).

En **HTTP**, les données transitent « en clair » : quiconque intercepterait les messages transitant sur le réseau, peut en lire le contenu. Ceci est évidemment problématique lorsque ces messages contiennent des mots de passe ou d'autres données personnelles.

En **HTTPS**, la connexion débute par un échange de clés entre le serveur et le navigateur. Les clés servent ensuite à crypter les messages avant leur envoi, puis les décrypter lors de la réception. Les sites web ne supportent pas tous ce protocole, car il nécessite une configuration spécifique sur le serveur et un certificat SSL.

Erreurs

Lors du chargement d'une ressource, différentes **erreurs** peuvent survenir. La plus répandue est l'**erreur 404** signalant une ressource manquante.

Chaque requête http comporte un **code d'état** :

- 200 : succès de la requête
- 301 ou 302 : redirections
- 401 : utilisateur non authentifié
- 403 : accès refusé
- 404 : fichier non trouvé
- 500, 503 et 504 : erreurs serveur.

Les URL

Chaque ressource disponible sur le Web (document HTML, image, séquence vidéo, programme, fichier, etc.) possède une adresse unique appelée **URL** (*Uniform Resource Locator*).

Certains caractères ne sont pas tolérés dans les URL et doivent être remplacés par des codes. Par exemple les espaces sont remplacés par %20, les guillemets par %22 et les pourcents par %25.

NB : La réécriture d'URL ou **URL rewriting** est une technologie serveur basée sur l'utilisation d'un fichier « .htaccess ». Elle permet de réécrire « proprement » certaines URL, ce qui explique les nombreuses URL sans extensions comme « <https://www.heh.be/accessibilite> ».

Il existe 2 façons de coder les URL : **URL absolues** et **URL relatives**.

Les URL absolues

Les **URL absolues** décrivent l'entièreté du chemin : du protocole jusqu'au nom du fichier et son extension, en passant par le nom du serveur et les noms des répertoires.

Dans la pratique, les URL absolues sont utilisées pour les ressources externes, provenant d'autres sites. Par exemple un lien depuis votre site vers une page Facebook ou un *iframe* chargeant une vidéo depuis YouTube.

Exemple :

<http://www.serveur.org/docs/index.htm>

Chaque URL absolue se compose :

- du protocole d'échange de données (HTTP, HTTPS, FTP, SMTP, etc.)
- du nom de domaine
- des répertoires d'accès du document
- du nom du document et de son extension

Les URL relatives

Les **URL relatives** sont plus simples à écrire car elles ne décrivent que le chemin entre le dossier où l'on se trouve et le fichier de destination.

Exemples de liens avec URL relatives pour accéder aux fichiers suivants :

C:/doc/page1.html
C:/doc/page2.html
C:/doc/dossier/page3.html
C:/page4.html

```
<a href="page2.html"> Lien vers page2 </a>
<a href="dossier/page3.html"> Lien vers page3 </a>
<a href="../page4.html"> Lien vers page4 </a>
```

Les URL relatives sont utilisées pour désigner tous les fichiers internes au site (sur un même domaine) et ce, pour deux raisons : cela simplifie le code (plus léger et plus lisible) et cela permet de transporter le site sans briser les liens internes. Effectivement, si vous changez votre site de dossier ou de domaine, les URL absolues des ressources internes s'en retrouvent modifiées.

Les ancrées



Une ancre est un lien pointant vers un endroit précis du document HTML. Les ancrées sont par exemple utilisées pour envoyer le visiteur en bas ou en haut de la page, ou au commencement d'un chapitre.

La cible, appelée aussi le point d'ancre, est désignée par la valeur de son attribut ID. Cette valeur peut s'utiliser en fin d'URL, précédée du symbole croisillon « # ».

```
<h2 id="chapitre3">Titre du chapitre 3</h2>
```

```
<a href="#chapitre3">Lire le chapitre 3</a>
```

1.10 Les images

Les formats d'images adaptés au Web

JPEG : format d'images matricielles, le JPEG est adapté aux images de type photographies (avec beaucoup de variation de couleur entre les pixels). Le JPEG applique une compression à taux variable, impliquant une perte de qualité souvent indiscernable à l'œil nu. Le JPEG ne permet pas la transparence.

Le JPEG est actuellement le format d'images le plus utilisé au monde et il serait temps de le remplacer par un de ses nombreux concurrents aux algorithmes de compression plus performants : **WEBP, JPEG 2000, AV1, HEIF, JPEG XR, JPEG XL**, etc.



PNG (Portable Network Graphic) : format d'images matricielles, alternative gratuite et plus performante au format GIF. Contrairement au JPEG, PNG applique une compression sans perte. PNG permet la translucidité (canal alpha) et la transparence.

Il existe 8 formats de PNG dont les 2 plus courants sont :

- **PNG 8 bits** : 256 couleurs (pour les images pauvres en couleur).
- **PNG 24 bits** : 16 777 216 couleurs disponibles : pour les images riches en couleur ou pour les photos (bien que la compression du JPEG rende les photos plus légères). Internet Explorer 6 (et les versions inférieures) ne gère pas la translucidité pour les PNG 24 bits.

GIF : format d'image qui permet la transparence et les animations (GIFs animés). Le format GIF a été mis au point par CompuServe en 1987 et est basé sur un algorithme de compression sans perte (si le nombre de couleurs est inférieur à 256) nettement plus performant que ceux des autres formats de l'époque (comme BMP).

Malgré un débat tendu sur la prononciation francophone de « gif », il s'avère finalement que la proconciation officielle est « jif ».

Entre 1994 et 2003, Unisys, détenteur du brevet de la compression LZW du GIF, réclamait des royalties aux auteurs de logiciels exploitant le format GIF.

WEBP : format d'image développé par Google et s'annonçant très prometteur : léger, avec ou sans perte lors de la compression (au choix), permet la transparence et les animations. Seules ombres au tableau : son incompatibilité avec IE et avec Safari pour les versions inférieures à 14.

SVG (Scalable Vector Graphics) : format d'image vectoriel basé sur le méta langage XML. Très léger, permet la translucidité et la transparence. Permet également des animations. Grâce à son format XML, les images SVG peuvent être modifiées par des scripts en temps réel.

Il existe de nombreux autres formats non adaptés au Web : **BMP, TIFF, PSD**, etc.

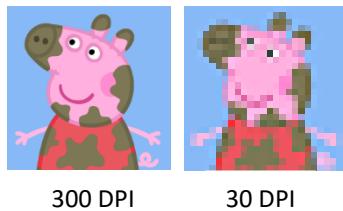
En conclusion, le format JPEG est un bon choix pour les photos et le format PNG 8 bits pour les images « synthétiques ». Le SVG reste un choix intéressant pour les logos, les images géométriques ou pour relever certains défis techniques.

Les formats de nouvelle génération comme WEBP sont la garantie de fichiers beaucoup plus légers. Mais ces formats entraînent malheureusement des problèmes de compatibilité encore à l'heure actuelle.

Les résolutions d'image sur le Web

Les fichiers images, comme les JPEG ou les PNG, sont constitués de points appelés « **pixels** ». Le nombre de pixels d'une image est appelé **définition** de l'image.

Lors de l'impression d'une image sur une certaine surface, on peut en mesurer le nombre de pixels par pouce (1 pouce = 2.54cm). Cette mesure s'appelle la **résolution de l'image** et est exprimée en **DPI** (*Dots Per Inch*), ou **PPP** en français (Points Par Pouce). Sur une même surface, plus la résolution est élevée, moins les points sont apparents, garantissant ainsi la netteté du résultat.



Une image imprimée en résolution de 300 DPI est d'une netteté irréprochable à l'œil nu. Cette résolution de 300 DPI est de facto conseillée pour les illustrations de livres ou de magazines. Si le support est destiné à être lu de loin, la résolution peut diminuer sans gêner le lecteur : 150 DPI pour un roll-up ou une affiche ; 50 DPI pour un panneau publicitaire à destination d'automobilistes ; etc.

Pour imprimer une image en 300 DPI (à l'aide d'une imprimante supportant cette résolution) sur une largeur de 5 cm (environ 2 pouces), il faut un fichier de 600 px de large : 2 pouces x 300 DPI = 600 px

La résolution d'une image est un paramètre du fichier image qui n'a pas d'impact sur le poids de l'image : le poids d'une image dépend de son nombre de pixels, pas de sa résolution.

Les logiciels de retouche d'images permettent de modifier la résolution. Si le poids de l'image varie lors de cette modification, c'est que le logiciel en a profité pour modifier la définition de l'image au passage.

Images	
ID de l'image	
Dimensions	256 x 256
Largeur	256 pixels
Hauteur	256 pixels
Résolution horizontale	96 ppp
Résolution verticale	96 ppp
Profondeur de couleur	24

Ces conseils en matière de DPI sont principalement destinés au domaine de l'impression. Ils concernent beaucoup moins le Web, voire pas du tout. Nous allons voir pourquoi...

Lors de l'ouverture d'un site dans un navigateur, par défaut, la taille d'affichage de l'image dépend seulement de la définition de l'image. Le paramètre résolution du fichier image n'intervient pas... et pour cause : le navigateur ignore la taille en pouces de l'écran du visiteur.

Pourtant, la netteté d'une image est tout aussi importante sur un écran que dans un magazine. Cette netteté à l'écran dépend de plusieurs facteurs, mais jamais du paramètre résolution du fichier image. Le premier facteur est l'écran : généralement entre 72 DPI et 108 DPI (ou PPI dans le cas d'un écran), selon les modèles et technologies d'écran. Le deuxième facteur sont les éventuelles dimensions spécifiées en HTML et en CSS qui peuvent étirer ou contracter l'image. Il est donc possible – et déconseillé – d'afficher une image de 200 x 200 px sur une zone de 400 x 400 px, provoquant une résolution 2 fois moindre que celle de l'écran du visiteur. L'effet inverse est toutefois impossible : afficher cette même image sur une zone de 100 x 100 px ne la rend pas plus nette : nous sommes limités par les pixels physiques de l'écran ! Exception faite de certaines technologies d'écran comme les écrans de mobiles...

Dans tous les cas, il est inutile de s'inquiéter du paramètre résolution d'une image destinée au Web.

Les images :

Pour afficher une **image** :

```

```

L'élément image possède de nombreux attributs :

alt Texte alternatif : description de l'image. **Cet attribut est obligatoire**, même s'il est laissé vide pour certaines images de décoration, n'illustrant pas le sujet.

src Source : URL de l'image. **Cet attribut est obligatoire**. Forcément.

height Dimensions : Hauteur et largeur. En règle générale, les dimensions des éléments HTML sont codées en CSS, mais une exception est prévue ici car préciser les dimensions d'une image permet au navigateur de réserver l'espace nécessaire avant leur chargement.

En HTML, les dimensions en pixels s'écrivent sans l'unité : **width="400"**

title Bulle d'informations lorsque l'on survole l'image sans bouger.

usemap Référence à une carte (voir « les cartes » plus loin).



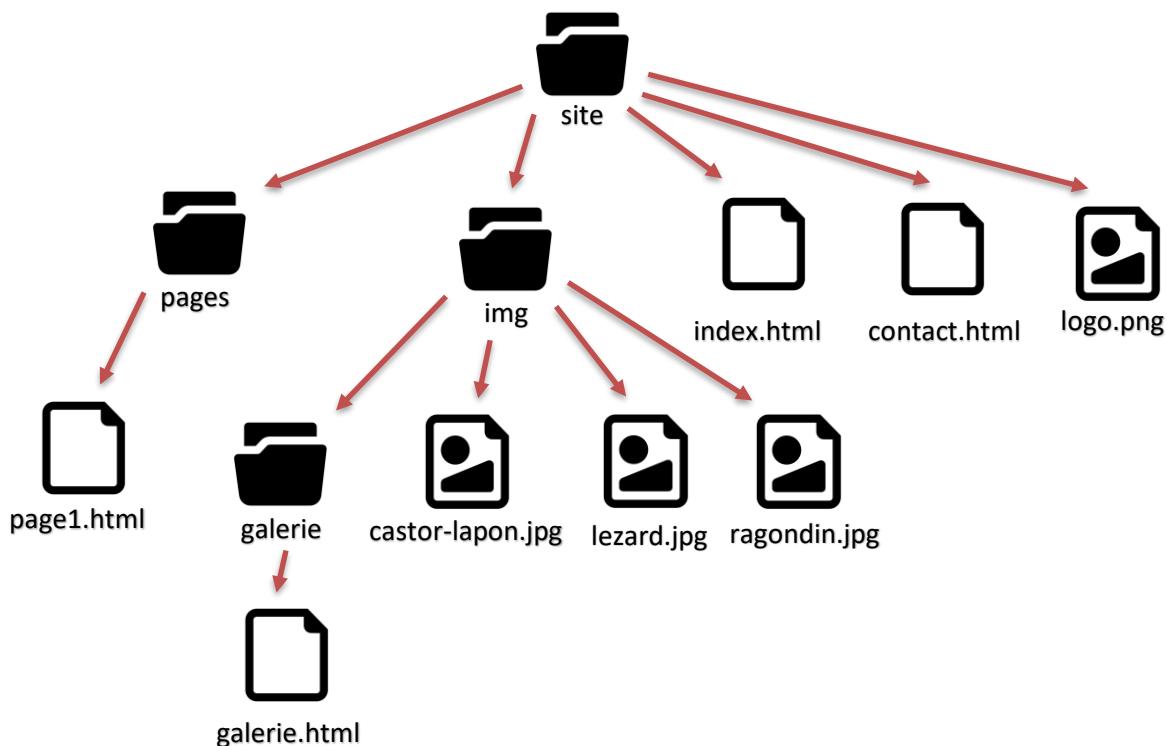
Accessibilité : Le critère WCAG 1.1.1 (A) recommande que toutes les images comportent un texte alternatif (attribut *alt*), sauf les images non porteuses d'informations (images décoratives).

Remarques :

- Le texte alternatif permet au visiteur de profiter des informations contenues dans les images s'il n'a pas accès aux images pour une raison technique ou un handicap visuel.
- En HTML, l'image ne fait pas partie de votre document. Après avoir chargé le fichier HTML, le navigateur charge les autres fichiers nécessaires dont les images.
- En conséquence, le fait d'utiliser la même image à plusieurs reprises dans un fichier HTML ne ralentit pas son chargement, car le navigateur ne la charge qu'une seule fois.
- Il est possible de choisir des valeurs de *width* et *height* différentes des dimensions de l'image, afin de la redimensionner. Si seule la largeur ou seule la hauteur est spécifiée, l'autre dimension s'adapte en gardant les proportions de l'image.

Exercice : les URL et les images

1. Commencez par créer cette arborescence de dossiers et de fichiers :



Remarques

- ✓ Par défaut, Windows cache les extensions de fichier connues au cas où un novice modifierait l'extension par mégarde. Modifiez ce comportement de Windows dans les paramètres d'affichage.
- ✓ Il est conseillé d'éviter les majuscules, les accents et les espaces (remplacés par des tirets ou des *underscore*) dans les noms de fichiers et de dossiers.
- ✓ Le choix des images n'a que peu d'importance pour cet exercice, mais veillez à ce que le poids et les dimensions soient raisonnables.
- ✓ Il est de bon usage de nommer la page d'accueil d'un site « index.html ».
- ✓ Dans la pratique, tous les fichiers HTML d'un site sont généralement placés à la racine (ici, le dossier « site ») mais pour compliquer notre exercice, des dossiers supplémentaires ont été créés.

2. Codez les fichiers HTML de façon que chaque fichier contienne un titre, une navigation dans une liste non-ordonnée et le logo. Vérifiez ensuite scrupuleusement chaque lien de cette navigation.
3. Faites-en sorte que le logo soit cliquable et envoie vers la page « index.html ».
4. Complétez la page « galerie.html » de façon qu'elle contienne toutes les images. Faites-en sorte que si l'on clique sur une image, cette image s'ouvre dans un nouvel onglet.
5. Faites-en sorte que le contenu de « galerie.html » soit suffisamment long pour provoquer l'apparition de la barre de scroll vertical (rajoutez du texte bidon ou d'autres images). Puis, rajoutez dans la navigation de chaque page, un lien pointant vers le bas de la page « galerie.html » (utilisation d'une ancre).
6. Vérifiez dans le Validator que vos pages ne contiennent pas d'erreur.

Légendes de médias

<figure> et **<figcaption>** permettent d'associer une légende à un contenu de type image, vidéo, tableau, etc.

```
<figure>
    
    <figcaption>Légende associée à l'image</figcaption>
</figure>
```

Remarques :

1. L'élément non obligatoire **<figcaption>** peut être codé avant ou après l'image.
2. On peut coder plusieurs images ou plusieurs autres médias dans un même élément **<figure>** de façon qu'ils aient la même légende.
3. Si le contenu de l'attribut **alt** est le même que celui de **figcaption**, l'attribut **alt** ne doit pas être codé.

Images adaptatives



L'élément **<picture>** est un conteneur définissant plusieurs sources à un élément ****. Le navigateur peut alors choisir la source répondant à diverses contraintes : les formats pris en charge, la densité de pixels ou les dimensions de l'affichage. Si aucune source ne répond aux contraintes, l'image conserve sa source (attribut **src**).

```
<picture>
    <source srcset="image.webp" type="image/webp">
    <source srcset="autre-image.jpg" media="(min-width:800px)">
    
</picture>
```

L'attribut **type** indique le type MIME du fichier fourni.

L'attribut **media** contient une condition sur le média visé.

Une autre solution consiste à employer les attributs **srcset** et **sizes** directement dans la balise ****.

```

```

En savoir plus ?

https://developer.mozilla.org/fr/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images

Les cartes



La balise **<map>** sert à définir plusieurs zones réactives (liens) sur une seule image. Elle permet à partir d'une seule image de proposer une série de différentes destinations internes comme externes. La relation entre une image et sa carte est précisée par l'attribut *usemap* dans l'élément **** :

```

```

Le conteneur **<map>** a pour seul attribut ***name*** : nom de la carte. Ce conteneur contient autant de balises **<area>** qu'il y a de zones réactives.

```
<map name="nom_de_la_carte">
    <area shape="..." coords="..." href="..." alt="...">
    <area shape="..." coords="..." href="..." alt="...">
</map>
```

Chaque zone réactive correspond donc à une balise **<area>** dont les attributs sont :

- ***shape*** : forme de la zone (cercle, rectangle ou polygone).
- ***coords*** : coordonnées de la zone (séparées par des virgules).
- ***href*** : adresse destination.
- ***alt*** : texte alternatif, obligatoire lorsque *href* est présent.
- ***title*** : commentaire qui apparaîtra lorsque le curseur de la souris passera sur la zone.

Zone circulaire

Une zone circulaire est définie par la balise **<area shape="circle">** et dont l'attribut ***coords*** prend pour valeurs l'abscisse et l'ordonnée du centre et le rayon du cercle.

```
<area shape="circle" coords="x,y,rayon" href="destination">
```

Zone rectangulaire

Une zone rectangulaire est définie par la balise **<area shape="rect">** et dont l'attribut ***coords*** prend pour valeurs les coordonnées des coins supérieur gauche et inférieur droit, séparées par des virgules.

```
<area shape="rect" coords="x1,y1,x2,y2" href="destination">
```

Zone polygonale

Une ne zone polygonale est définie par la balise **<area shape="poly">** et dont l'attribut ***coords*** prend pour valeurs la liste des abscisses et ordonnées de chacun des sommets jusqu'à revenir aux coordonnées du sommet de départ. Le choix du sommet de départ est libre, mais il est important de coder les sommets suivants dans l'ordre où ils se succèdent sur le périmètre de la forme.

```
<area shape="poly" coords="x1,y1,x2,y2,x3,y3,...,x39,y39,x1,y1"
      href="destination">
```

Exemple d'une image de pizza réactive :

L'image, comme toujours, est rectangulaire, mais seule la pizza (cercle au centre de l'image) est cliquable.

```

<map name="carte_pizza">
    <area shape="circle" coords="75,75,75" href="commande.htm">
</map>
```

Les icônes avec *Font Awesome*



Ce n'est pas évident d'inclure proprement des icônes dans les sites web tout en les alignant sur le texte. Une solution répandue consiste à utiliser une police d'icônes.

Parmi ces polices d'icônes, *Font Awesome* est une des plus célèbre par son intégration à Bootstrap. Sa version gratuite compte actuellement plus de 1500 icônes. L'inclusion d'un fichier CSS, permet ensuite d'appliquer la police aux éléments de classe « fa ». Une seconde classe « fa-heart », « fa-car », etc. permet de choisir l'icône souhaitée :

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

<i class="fa fa-heart"></i>
<i class="fa fa-car"></i>
```

Documentation complète : <https://www.w3schools.com/icons/default.asp>

La Favicon



La Favicon est l'icône de la page, visible dans l'onglet. Ses dimensions doivent être de 16x16 ou de 32x32. Son format est idéalement « .ico » ou « .png », même si d'autres formats et tailles sont supportés, selon les navigateurs.

```
<link rel="icon" type="image/png" href="/images/mafavicon.png">
```

Avec l'arrivée des smartphones, de nombreuses autres dimensions d'icônes de site ont vu le jour car les icônes se retrouvent dorénavant sur les bureaux, sur les écrans d'accueil, etc. Voici un code plus complet proposant une icône 16x16, une 32x32, une 144x144 et une 180x180 en plus de couleurs personnalisant l'interface du navigateur :

```
<link rel="apple-touch-icon" type="image/png" href="logo180.png" sizes="180x180">
<link rel="icon" type="image/png" href="logo32.png" sizes="32x32">
<link rel="shortcut icon" type="image/png" href="logo16.png">
<meta name="msapplication-TileImage" content="logo144.png">
<meta name="msapplication-navbutton-color" content="#505053">
<meta name="msapplication-TextColor" content="#505053">
<meta name="theme-color" content="#505053">
```

Les sons et les vidéos

Contrairement à l'insertion d'images, l'insertion des sons et des vidéos a toujours été problématique dans le monde du Web. Ceci en grande partie à cause du nombre de formats différents, des différences de comportement des navigateurs et également à cause du fait que le traitement de ces sons et de ces vidéos n'est pas opéré par les navigateurs mais par des lecteurs/plug-ins additionnels.

Parmi les anciennes solutions pour l'insertion de sons et de vidéos dans les pages, on retrouve la balise **<embed>** (non reconnue comme spécification HTML), la balise **<object>** ou l'utilisation de technologies comme le Flash.

Heureusement, HTML5 apporte une lueur d'espoir avec les éléments **<video>** et **<audio>**.

```
<video controls src="video.mp4" preload="auto" autoplay muted loop>
Description textuelle alternative</video>
```

- L'attribut **controls** ou **controls="controls"** équipe le lecteur de boutons de contrôle (play, pause, etc.)
- L'attribut **preload="auto"** permet de spécifier au navigateur de débuter le téléchargement de la vidéo tout de suite, en anticipant le fait que l'utilisateur lira la vidéo. Attention de ne pas abuser des préchargements de vidéo : il est préférable de ne les précharger que si elles sont incontournables.
- L'attribut **autoplay** ou **autoplay="true"**, permet de lancer la lecture automatiquement. Cela peut également être problématique avec une connexion à faible bande passante ou sur un terminal mobile. De manière générale, évitez d'imposer vos choix à l'utilisateur... et à sa connexion internet. D'ailleurs certains navigateurs bloquent ce paramètre s'il n'est pas accompagné de **muted**.
- L'attribut **muted** ou **muted="true"** coupe le son de la vidéo.
- L'attribut **loop** ou **loop="true"** provoque une lecture en boucle de la vidéo.

Variante avec plusieurs sources de formats différents :

```
<video width="400" height="220" controls="controls">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <source src="video.ogv" type="video/ogg">
  Ici l'alternative à la vidéo : un lien, un message, etc.
</video>
```

	H.264/MP4	OGG Theora	WebM
	Firefox 34+	Firefox 3.5+	Firefox 4+
	Opera 25+	Opera 10.5+	Opera 10.6+
	Internet Explorer 9+	non	si codecs installés
	Google Chrome 4+	Google Chrome 4+	Google Chrome 6+
	Safari 5+	non	non



Accessibilité : Les critères WCAG 1.2.1 à 1.2.5 (A et AA) recommandent des retranscriptions textuelles, des audiodescriptions et des sous-titres synchronisés pour la plupart des contenus audio et vidéo.

1.11 Les *frame* et *iframe*

<frame>

Technologie aujourd’hui obsolète, les *frames* consistaient à séparer une page en cadres et à charger des portions de code HTML dans chaque cadre. Cette technologie présentait principalement 2 avantages :

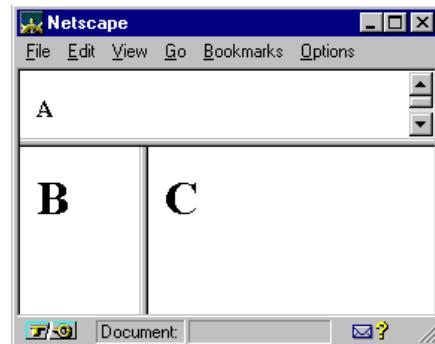
- Les parties communes à plusieurs pages comme la navigation, le pied de page, etc. n’étaient codées qu’une seule fois.
- La navigation au sein du site provoquait des chargements allégés puisque d’une page à l’autre, seuls certains cadres étaient rechargés, pas tous. Cet intérêt est moindre depuis que les navigateurs exploitent un cache : les fichiers (css, images, vidéos, etc.) ne sont plus rechargés à chaque page.

Cependant, les *frames* présentent aussi quelques inconvénients :

- Un manque cruel d’esthétisme (bien que les bordures et espaces puissent être enlevés avec les attributs : border="0", frameborder="no" et framespacing="0")
- Une faiblesse de référencement (la page principale est dénuée de contenus, ce qui pénalise le positionnement du site)

Exemple :

```
<html>
<head></head>
<frameset rows="30%,70%">
  <frame src="A.htm" name="a">
  <frameset cols="30%,70%">
    <frame src="B.htm" name="b">
    <frame src="C.htm" name="c">
  </frameset>
</frameset>
</html>
```



Dans la page principale, la balise *frameset* remplace la balise *body*. Elle sert à séparer la page en différents frames et définit les positions et répartitions de ces frames.

Dans cet exemple, la page est séparée une première fois en deux lignes. La première ligne charge le contenu de la page « A.htm ». La seconde ligne est séparée en deux colonnes dont la première charge le contenu de la page « B.htm » et la seconde le contenu de la page « C.htm ».

Les ascenseurs (ou barre de défilement), apparaissent automatiquement. Mais vous pouvez tout de même indiquer si la fenêtre doit ou non posséder une barre de défilement avec l’attribut *scrolling="no"*

L’attribut *name* indique le nom du cadre afin que ce *frame* puisse être utilisé comme cible d’un lien hypertexte (par exemple *target="c"*). Par exemple si un visiteur clique sur un lien dans le frame B pour charger un contenu dans le frame C.

<iframe>

Si les balises **<frame>** et **<frameset>** sont dorénavant dépréciées, la balise **<iframe>** est encore utilisée pour importer des contenus HTML externes au sein d’une page. Elle est, par exemple, souvent utilisée pour importer des contenus de YouTube, de Facebook, etc.

```
<iframe src="source.html" height="400" width="600"></iframe>
```

1.12 Les formulaires

Un site qui vit est un site en dynamique avec le visiteur. Les formulaires répondent à ce besoin en permettant de recueillir les informations entrées par les internautes.

Les formulaires peuvent prendre différentes formes : formulaire d'inscription ou de connexion, formulaire de commande dans un e-commerce ou formulaire de commentaire dans un forum ou un blog, ou encore simple formulaire de recherche.

La structure du formulaire (les différents champs à remplir) se fait en HTML. Mais le traitement des informations (leur affichage, leur enregistrement dans une base de données, etc.) nécessite un langage dynamique comme le PHP ou le JavaScript.

Formulaire : <form>

```
<form action="index.php" method="post"> ... </form>
```

Le formulaire est délimité dans le code HTML par : <form> </form>. Tous les champs d'un même formulaire doivent être contenus entre ces 2 balises.

Attention, l'élément <form> est un bloc qui ne peut avoir pour enfants que les éléments : <p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <div>, <pre>, <address>, <fieldset>, <ins> et .

action	Indique la destination des données. Peut contenir l'URL du programme qui traite les données ou l'adresse email de destination (exemple : <i>action="mailto:ivan.miller@heh.be"</i>).
method	Deux valeurs possibles : get ou post . Avec <i>get</i> , les données apparaissent dans l'URL de la page, laissant la possibilité à l'utilisateur de les modifier. Cette méthode convient aux formulaires de recherche. Avec <i>post</i> , les données sont contenues dans l'en-tête de la page de manière invisible pour l'utilisateur. Cette méthode convient davantage aux formulaires de connexion, de commandes, etc.
autocomplete	Autorise le navigateur à autocompléter les champs. Peut valoir « on » (par défaut) ou « off ». Peut être utilisé dans l'élément <form> ou dans d'autres éléments de formulaire (<input>, <select>, <textarea>, etc.)
enctype	Format d'envoi des données (uniquement utile si le formulaire permet l'envoi de fichiers).

Conteneur : <fieldset>

L'élément `<fieldset>` est un bloc dont l'usage est conseillé dans le cas des formulaires afin de contenir les champs.

L'élément `<legend>`, enfant de `<fieldset>`, permet d'ajouter une légende apparaissant par défaut en haut à gauche à mi-hauteur de la bordure.

```
<form action="..." method="...">
  <fieldset>
    <legend> Identité </legend>
    ...
  </fieldset>
</form>
```

Champs de saisie : <input>

Les formulaires comportent de nombreux champs de saisie dont la nature peut varier : champ texte, champ mot de passe, bouton, case à cocher, etc.

Tous ces champs de saisie se codent avec l'élément `<input>`, dont l'attribut `type` définit la nature du champ. Si le navigateur ne comprend pas la valeur de l'attribut `type`, il affiche un champ texte.

L'élément `<input>` propose de nombreux **attributs** :

- **`name`** : le nom du champ. Indispensable pour récupérer les données côté serveur.
- **`value`** : la valeur du champ, parfois visible (dans un champ texte ou intitulé sur un bouton), parfois invisible (case à cocher). Lors de l'envoi des données au serveur, celui-ci reçoit des associations `name/value`.
- **`placeholder`** : texte indicatif dans le champ, disparaissant lorsqu'on complète le champ.
- **`required`** : champ obligatoire (la soumission du formulaire est bloquée si le champ est vide).
- **`readonly`** : le champ se trouve alors en mode lecture seule, c'est-à-dire que le texte qui s'y trouve ne peut pas être effacé ou modifié.
- **`disabled`** : désactive la balise. Cet attribut ne prend pas de valeur en HTML5.
- **`autocomplete`** : permet au navigateur d'autocompléter le champ. Peut valoir « `on` » (par défaut) ou « `off` ».
- **`list`** : la référence à une `datalist` (liste d'auto-complétion)
- **`pattern`** : spécifie quelles formes de réponses sont acceptées.
- **`size`** : la taille du champ en nombre de caractères.
- **`maxlength`** et **`minlength`** : en nombre de caractères, la taille maximale et minimale de la valeur.
- **`max`** et **`min`** : seulement pour les champs de type date et nombre.
- **`step`** : l'écart (par exemple 1000 ou 0.1), seulement pour les champs nombres.
- **`src`** : la source, seulement pour les champs de type image.
- **`type`** : les nombreuses possibilités de champs de saisie (listées et expliquées plus loin) parmi lesquelles `text`, `submit`, `password`, `radio`, `checkbox`.



Accessibilité : Le critère WCAG 1.1.1 (A) recommande que les boutons de formulaire aient des attributs `value` descriptifs. Par exemple « Connexion » ou « Passer la commande » plutôt que « Ok »...

Différents types de champs de saisie :

- <input type="button"> : bouton neutre (destiné sans doute à une fonction JavaScript)
- <input type="checkbox"> : case à cocher
- <input type="color"> : choix de couleur via une boîte *color-picker*
- <input type="date"> : champ date
- <input type="datetime-local"> : champ date et heure
- <input type="email"> : champ email
- <input type="file"> : champ fichier joint
- <input type="hidden"> : champ caché (donnée pour le serveur invisible au visiteur)
- <input type="image"> : champ image
- <input type="month"> : champ mois
- <input type="number"> : champ nombre (entier ou réel)
- <input type="password"> : champ mot de passe (les caractères sont remplacés par des *)
- <input type="radio"> : bouton radio
- <input type="range"> : intervalle avec curseur
- <input type="reset"> : champ de réinitialisation du formulaire (peu utile)
- <input type="search"> : champ recherche
- <input type="submit"> : champ de soumission/validation du formulaire (indispensable)
- <input type="tel"> : champ numéro de téléphone
- <input type="text"> : champ texte (par défaut)
- <input type="time"> : champ heure
- <input type="url"> : champ url
- <input type="week"> : champ semaine

Exemples

Les champs les plus courants sont les *text*, *password*, *radio*, *checkbox*, *number* et *submit*. Voici un exemple de chaque :

Un exemple de champ *text* :

```
<input type="text" name="login" maxlength="32" required>
```

Un exemple de champ *mot de passe* :

```
<input type="password" name="mdp" maxlength="32" required>
```

Deux boutons *radio* :

<input type="radio" name="titre" value="M." checked> Monsieur	<input checked="" type="radio"/>
<input type="radio" name="titre" value="Mme"> Madame	<input type="radio"/>

NB : l'attribut *name* prend la même valeur pour que le navigateur comprenne le lien entre les boutons. Un seul choix peut être effectué parmi les boutons radio de même *name*.

Deux boutons *checkbox* :

<input type="checkbox" name="sport[]" value="judo"> Judo	<input checked="" type="checkbox"/>
<input type="checkbox" name="sport[]" value="golf"> Golf	<input type="checkbox"/>

NB : l'attribut *name* prend la même valeur, mais cette fois on peut cocher plusieurs choix. Le serveur est donc susceptible de recevoir plusieurs valeurs, et c'est pour cela que l'on donne une syntaxe de tableau au *name* en le pourvoyant de crochets.

Un exemple de champ *nombre* :

```
<input type="number" name="poids" step="0.1" min="0" max="1000">
```

Un exemple de bouton de soumission :

```
<input type="submit" name="ok" value="Ok">
```

Ok

Zone cliquable étendue : <label>

Les champs de formulaires sont cliquables, parfois pour provoquer le focus (le curseur y clignote, invitant l'utilisateur à taper le contenu), parfois pour cocher un bouton *radio* ou *checkbox*. De nombreux utilisateurs ont l'habitude de cliquer sur l'intitulé du champ plutôt que sur le champ lui-même. Cette fonctionnalité n'est présente que si vous avez étendu la zone cliquable en codant un élément <**label**>.

Il est conseillé de toujours coder un <**label**> pour les champs de saisie suivant : *text*, *checkbox*, *radio*, *file* et *password*.

Le lien entre un <**label**> et son champ de formulaire est exprimé par l'attribut *for* qui reprend la valeur de l'*id* du champ.

Exemple de label :

```
<label for="pwd">Pwd : <input type="password" id="pwd"></label>
```

Autre exemple de label :

```
<p><label for="pwd">Pwd :</label></p>
<p><input type="password" id="pwd"></p>
```



Accessibilité : Le critère WCAG 1.1.1 (A) recommande que chaque champ de formulaire soit lié à une étiquette par l'utilisation d'un élément <**label**>.

Champ multi-lignes : <textarea>

L'élément <**textarea**> permet au visiteur d'introduire un texte long, de plusieurs lignes.

Attributs :

- ***name*** : le nom du champ.
- ***rows*** : le nombre de lignes.
- ***cols*** : le nombre de colonnes (nombre de caractères par lignes).
- ***readonly*** : le champ se trouve alors en mode lecture seule, c'est-à-dire que le texte qui s'y trouve ne peut pas être effacé ou modifié.

Exemple de champ multi-lignes :

```
<textarea name="commentaire" rows="6" cols="100"></textarea>
```

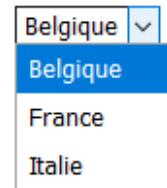
Dépourvue d'attribut *value*, l'élément <**textarea**> a pour valeur son contenu. À l'instar de l'élément <**pre**>, <**textarea**> applique une police mono-espacée à son contenu et interprète tous les espaces et sauts de lignes qui s'y trouvent. C'est pourquoi, on veille à ne pas espacer la balise ouvrante et la balise fermante de cet élément.

Sitôt que le texte est plus grand que la zone de texte, des barres de défilement apparaissent.

Listes déroulantes : <select>

Plus pratique que les boutons radio et checkbox lorsque le nombre de possibilité est élevé, par exemple pour le choix du pays ou de la langue, mieux vaut utiliser les listes déroulantes **<select>** dont les attributs sont :

- **name** : le nom de la liste.
- **size** : (facultatif) dont la valeur unité (par défaut) signifie qu'on a affaire à une liste déroulante sinon c'est une liste normale mais avec barre de défilement.
- **multiple** : (facultatif) qui signifie qu'il est possible de sélectionner plusieurs valeurs. Si *multiple* n'est pas spécifié, il ne sera possible de faire qu'un seul et unique choix.



Le conteneur **<select>** contient autant d'éléments **<option>** que de choix possibles. Chaque **<option>** comprend un attribut *value*. C'est la valeur de l'option sélectionnée qui est envoyée au serveur lors de la soumission du formulaire.

Exemple :

Dans quel pays habitez-vous ?

```
<select name="pays">
    <option value="B" selected>Belgique</option>
    <option value="F">France</option>
    <option value="I">Italie</option>
</select>
```

Il est également possible de créer des groupes au sein de la liste :

```
<select name="pays">
    <optgroup label="Europe">
        <option value="B">Belgique</option>
        <option value="F">France</option>
        <option value="I">Italie</option>
    </optgroup>
    <optgroup label="Asie">
        <option value="J">Japon</option>
    </optgroup>
</select>
```

Liste d'autocomplétion : <datalist>

<datalist> permet de lier un champ de formulaire à une liste prédéfinie, pour que le navigateur suggère les valeurs de la liste.

Votre animal de compagnie préféré :

```
<input list="animaux" type="text">
<datalist id="animaux">
    <option value="chat">
    <option value="cobaye">
    <option value="cochon">
    <option value="hamster">
</datalist>
```

<datagrid> est une sorte de **<datalist>** plus complexe permettant de structurer les informations sous formes d'arbres (hiérarchie).

Fichiers joints



La balise `<input type="file">` permet d'envoyer un fichier via un formulaire. Ce champ comporte un bouton qui permet l'ouverture d'une boîte de dialogue de choix du fichier.

Aucun fichier sélectionné.

Point capital nécessaire à l'envoi d'un fichier en pièce jointe : il faut changer la valeur de l'attribut **enctype** du conteneur **form** en multipart/form-data. En effet, **enctype** précise au navigateur le type MIME du message envoyé lors de la validation du formulaire. Le type MIME plain/text signifie texte brut, alors que multipart/form-data signifie que le message peut contenir des données binaires.

Exemple :

```
<form action="mailto:votre@email" method="post" enctype="multipart/form-data">
  <p>
    <input type="file" name="nom_du_champ" size="taille">
  </p>
</form>
```

Champ résultat : `<output>`



L'élément `<output>` est destiné à contenir le résultat d'un calcul (souvent codé en JavaScript), par exemple le montant total.

Exemple :

```
<input type="range" id="a" value="50">
+ <input type="number" id="b" value="25">
= <output name="x" for="a b"></output>
```

Exercice : formulaire

Commande de pizza

Nom

Prénom

Pizza

Suppléments
 Fromage
 Olives
 Poivrons
 Jambon

Livraison
 Au restaurant
 Livraison à Domicile :

Payement
 Paypal Visa Cash

Commander la pizza !

1.13 Les balises *meta*

Les balises meta sont toujours codées dans le `<head>` car elles contiennent des informations (ou métadonnées) qui ne sont pas affichées. Ces métadonnées sont utiles aux moteurs de recherche, aux navigateurs et aux visiteurs curieux.

1. Spécifier le **format d'encodage** (cette information est obligatoire) :

```
<meta charset="utf-8">
```

2. Spécifier **l'auteur du site** :

```
<meta name="author" content="Prénom NOM">
```

3. Spécifier le **viewport** (pour adapter l'affichage aux mobiles, si le CSS du site est prévu pour) :

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

4. Spécifier **une description** : il est conseillé de choisir une description de 200 à 300 caractères.

Cette description sera affichée sous le titre de votre site dans les résultats des moteurs de recherche. La description est indispensable à un bon référencement et sera de préférence différente sur chaque page de votre site.

```
<meta name="description" content="...">>
```

5. Spécifier **des mots-clés** : les mots-clés sont utilisés par le moteur de recherche Bing (pas par Google et Yahoo). Il est conseillé de choisir des mots-clés généraux et des mots-clés précis. Il n'est pas indispensable de décliner les mots (pluriel, féminin, etc.).

```
<meta name="keywords" content="Mot-clé1, Mot-clé2, Mot-clé3">
```

6. Spécifier les **outils de développement** du site tels que les logiciels de traitement d'images, les éditeurs, les CMS, etc. avec la méta **generator** :

```
<meta name="generator" content="logiciel1, logiciel2">
```

7. Choisir le comportement des **robots** : Les « robots » sont des logiciels envoyés par les moteurs de recherche afin de sonder les pages de vos sites. Il est possible de permettre ou pas l'indexation (**index** ou **noindex**), de permettre ou pas l'exploration des liens (**follow** ou **nofollow**). Si cette balise n'est pas spécifiée, les valeurs par défaut sont *index* et *follow*. La valeur *Noimageindex* empêche le référencement des images, et *freesurvey* empêche de suivre les liens pointant vers les images. Vous pouvez bien entendu combiner ces valeurs.

```
<meta name="robots" content="noindex,nofollow">
```

8. **Le rafraîchissement et la redirection** : Bien que ce ne soit pas conseillé, il est possible de faire recharger une page périodiquement. Ainsi, il est possible d'ordonner au navigateur de recharger une page toutes les **n** secondes grâce à la méta **Refresh** :

```
<meta http-equiv="refresh" content="n">
```

On peut même stipuler le chargement d'une page différente, très utilisée lorsqu'on change d'hébergeur afin de rediriger le navigateur vers un autre site :

```
<meta http-equiv="refresh" content="n" URL="...">>
```

En conclusion, à part la méta-balise précisant le format d'encodage des caractères qui est obligatoire, il revient au développeur de choisir les informations qu'il veut stipuler dans ses pages.

Viewport, device-width et initial-scale

Spécifier le **viewport** (pour permettre un affichage adapté aux mobiles) :

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Sur mobiles, les pixels physiques (**surface physique**) diffèrent souvent des pixels CSS (**surface utilisable** ou **device-width**). Prenons l'iPhone 4 et sa définition physique de 640x960px pour une définition CSS de 320x480px. On dit alors que le **pixel ratio** est de 2.0 (= 640/320).

La taille du **viewport**, elle, ne dépend d'aucune des deux surfaces précédentes, mais dépend du navigateur. Souvent, il s'agit d'une dimension élevée (souvent entre 800px et 1024px) afin de pouvoir y afficher des sites web en entier mais dézoomés.

Le niveau de zoom initial (**initial-scale**) vaut *device-width / viewport*. Par exemple, Safari mobile et son *viewport* de 980px affiché sur un iPhone 4 de surface physique 640px mais de surface utilisable 320px, applique un zoom de 0.3265x (= 320/980). En forçant un zoom initial de 1.0 dans le *viewport*, le navigateur affichera le site sur 320px et pas sur 980px et ce sera au CSS d'adapter le site à cette largeur de 320px. C'est l'utilité de la méta-balise ci-dessus.

Exercice :

Quel est le zoom initial sur un Samsung S5 de 1080x1920 px avec un pixel ratio de 3.0, utilisant Android Browser 3 dont le *viewport* est de 800px ?

Exercice : Analyse de métabalisées

Rendez-vous sur les sites <http://www.bigmoustache.com/> et www.moustachebikes.com afin d'en analyser les métabalisées.

Dans chaque site, accédez au code HTML (Ctrl+u dans certains navigateurs, ou clic-droit -> voir la source dans d'autres). Relevez toutes les métabalisées citées dans ce chapitre et retranscrivez (en français, pas en code source) ci-dessous les informations qu'elles contiennent.

Exemple :

La page d'accueil du site www.mons.be contient les informations suivantes sous la forme de métabalisées :

- Le format d'encodage utilisé est UTF-8
- Le logiciel Plone a été utilisé pour la création du site. Plone est un CMS.
- La description de la page d'accueil est la suivante : « Actualités, événements, services en ligne, photos, vidéos... Tout sur Mons, capitale européenne de la culture en 2015 ! »
- Parmi les métabalisées non vues dans ce chapitre, on retrouve « imagetoolbar »

À votre tour !

Exercice final HTML : trouvez les erreurs W3C

Le code HTML suivant contient **10 erreurs** (des erreurs qui seraient signalées par le W3C). Trouvez-les et corrigez-les !

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
    <title>Stan Lee</title>
<body>
    <span>
        <h1>À propos des Comics</h1>
    </span>
    <article id="stanlee" class="article">
        <h2 id="stanlee">Stan Lee</h2>
        <p>Stanley Lieber, dit <strong>Stan Lee</strong>, né le 28
décembre 1922 à New York et mort le 12 novembre 2018 à Los Angeles, est un
scénariste américain et éditeur de comics. <br>Son nom est associé à <a
href="#marvel">Marvel Comics</a>, pour lequel il a imaginé des super-
héros : </p>
        <ul>
            <h3>super-héros</h3>
            <li>Spider-Man</li>
            <li>Hulk</li>
            <li>Iron Man</li>
            <li>les X-Men</li>
            <li>ainsi que bien d'autres personnages...</li>
        </ul>
        <p class="illustration"><img="img/super-heros.jpg"></p>
    </article>

    <article id="marvel" class="article">
        <h2>Marvel Comics</h2>
        <p>Dans les années 1970, <a href="#stanlee">Stan Lee</a>
abandonne progressivement l'écriture de scénarios et la gestion éditoriale
des revues. Confiant ses personnages à une équipe de jeunes auteurs qu'il a
formés (Roy Thomas, Gerry Conway, Len Wein...), il se consacre à la tournée
des universités ou aux relations avec la presse, afin de faire connaître
l'univers <b>Marvel</b>. </p>
    </article>
</body>
</html>
```

Chapitre 2 : CSS



Chapitre 2 : CSS

2.1 Introduction aux feuilles de style CSS

Le langage CSS (*Cascading Style Sheets* ou feuilles de style en cascade) est utilisé pour définir la mise en page des documents HTML.

L'utilisation de CSS implique une séparation stricte entre structure (HTML) et présentation (CSS) : toutes les instructions de présentation (couleurs, polices, dimensions, positions, etc.) sont confinées.

Avant l'arrivée des CSS en 1996, les informations de mise en page étaient implémentées en HTML et étaient affreusement redondantes : la police, la taille, la couleur, étaient répétées pour chaque paragraphe et chaque titre. Ce n'est plus le cas avec les CSS.

```
<body bgcolor="pink" color="white">
<p><font family="Impact" size="24" color="purple">Texte</font></p>
<p><font family="Impact" size="24" color="purple">Texte</font></p>
<p><font family="Impact" size="24" color="purple">Texte</font></p>
</body>
```



```
body {
    background-color: pink;
    color: white;
}
p {
    font-family: Impact, sans-serif;
    font-size: 1.5rem;
    color: purple;
}
```

Les avantages apportés par le CSS sont multiples :

- ✓ Simplifie le code HTML, donc le rend plus léger et plus lisible.
- ✓ De facto, le chargement des pages en est accéléré.
- ✓ Facilite la maintenance du code.
- ✓ Garantit l'homogénéité de la mise en page.
- ✓ Améliore le positionnement des sites dans les moteurs de recherche.
- ✓ Permet un affichage différent selon les périphériques de sortie.
- ✓ Permet de nouvelles possibilités de mise en page comme le positionnement des blocs, les ombres, les transitions, les transformations, les animations, les filtres, etc.

Les CSS permettent de réduire la taille des documents HTML. De nombreuses études ont montré que l'utilisation des CSS, réduit la taille des documents HTML de moitié. Ce qui se perd au niveau du document HTML se récupère toutefois par le poids du fichier CSS, mais celui-ci ne fera que 25 à 35% de la page d'origine. Ce qui permet finalement un gain de grosso modo 15 à 25%.

2.2 Comment utiliser les CSS

Le principe d'une feuille de style est d'associer des propriétés visuelles à des éléments HTML.

Exemple appliquant à tous les éléments `` une couleur de texte verte :

```
strong { color:green; }
```

Vous avez probablement remarqué que votre navigateur applique du gras à tous les éléments ``. C'est parce que le navigateur possède ses propres feuilles de style par défaut.

Pour en revenir à l'HTML, Le choix des balises doit toujours être guidé par leur sémantique ou leur fonctionnalité, jamais par leur apparence puisque celle-ci est modifiable en CSS. Par exemple si vous souhaitez agrandir la taille d'un paragraphe, n'utilisez pas `<h1>`, laissez l'élément `<p>` et ajoutez une déclaration CSS.

L'HTML fournit expressément deux éléments neutres : le bloc `<div>` et la ligne ``, qui n'ont aucune propriété visuelle par défaut, aucune sémantique et servent précisément à être « habillés » par le CSS.

3 différents types de feuilles de style

1. Styles incorporés
2. Feuilles de style incorporées
3. Feuilles de style externes

1. Styles incorporés : sous la forme d'un attribut HTML « `style` ». Cette écriture est tolérée mais va tout de même à l'encontre de la séparation entre structure et présentation.

Exemple :

```
<p style="color:blue;">Bonjour</p>
```

2. Feuilles de style incorporées : sous la forme d'un élément `<style>` situé dans l'élément `<head>`. Cette écriture est utile par exemple dans le cas d'une page sensiblement différente des autres, qui mériterait quelques lignes de CSS exclusives.

Exemple :

```
<style>
a { color:blue; }
</style>
```

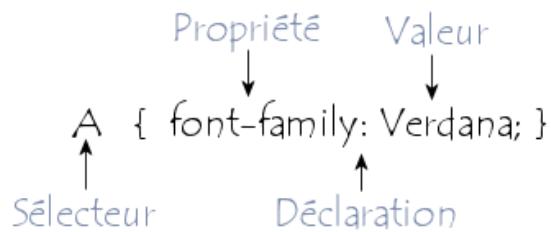
3. Feuilles de style externes : sous la forme d'un fichier lié aux pages HTML par l'élément `<link>`, toujours situé dans le `<head>`. Ces feuilles externes sont les plus utiles : une seule feuille de style peut être liées à toutes les pages du site et garantir un maximum d'avantages.

Exemple :

```
<link rel="stylesheet" href="fichier.css">
```

Avant l'HTML5, dans les balises `<link>` et `<style>`, un attribut obligatoire `type="text/css"` indiquait qu'il s'agissait de CSS. Cet attribut n'est plus obligatoire depuis.

Comment écrire une feuille de style



Sélecteur : définit quels éléments sont concernés par les déclarations qui suivent.

Déclaration : le bloc de déclarations compris entre { } peut contenir plusieurs déclarations séparées par des « ; ». Chaque déclaration est une paire propriété / valeur.

Propriété : décrit un aspect de la présentation d'un élément : comme sa taille ou sa couleur.

Valeur : définit la valeur de la propriété. Ci-dessus, la propriété « font-family » (la police d'écriture) est « Verdana ».

Règle : le tout s'appelle une règle CSS

Exemple : voici comment écrire que les éléments `` doivent s'afficher en gras et en rouge :

```

strong {
  font-weight: bold;
  color: red;
}
  
```

C comme Cascade

Pour rappel, l'abréviation CSS signifie *Cascading Style Sheets*. Le terme *Cascading* souligne une propriété importante du langage : **les caractéristiques de présentation se propagent « en cascade » d'un élément à ses descendants**.

Par exemple la déclaration `color:blue` appliquée à un `<article>`, applique également la couleur bleu à tous les descendants de cet article. Toutefois, si une autre couleur est appliquée à un des descendants, c'est cette nouvelle couleur qui s'applique à cet élément et à ses propres descendants.

Certaines propriétés CSS ne s'héritent pas : les positions, les marges, les dimensions (largeur et hauteur), les bordures, etc.

Commentaires en CSS

Commentaires : Il est possible de coder des commentaires presque n'importe où dans une feuille de style CSS en les entourant de /* et */, comme en langage C.

Les médias

Il est possible de tester le périphérique de sortie, afin d'appliquer ou non des règles CSS, par exemple pour corriger l'apparence du site lors de son affichage sur un mobile ou lors de son impression.

Les périphériques (ou médias) que l'on peut cibler sont : **screen** (écrans), **handheld** (Périphériques mobiles ou de petite taille), **print** (impression), **aural** ou **speech** (synthétiseurs vocaux), **braille** (plages braille), **embossed** (imprimantes braille), **projection** (projecteurs), **tty** (terminal avec police mono-espacée), **tv** (téléviseur), **all** (tout).

Ces valeurs peuvent être utilisées en valeur de l'attribut *media* dans les balises <link> ou <style> :

```
<link rel="stylesheet" href="style.css" media="all">
```

Ces valeurs peuvent aussi être utilisées dans le code CSS avec la commande @media :

```
@media print {  
    #entete, #banniere, #pied {  
        display:none ;  
    }  
}
```

Avec le **CSS3**, apparaît une façon plus précise de définir les médias concernés : les **media-queries**. C'est cette technologie qui est au cœur du **Responsive Design** : pratique visant à rendre les sites web capables de s'adapter à toutes les définitions d'écran !

Les **media-queries** sont composées d'expressions booléennes (toujours entre parenthèses) et de médias (*screen*, *all*, ...) reliés par des opérateurs logiques : **and** (et), **only** (uniquement), **not** (non). L'opération « ou » s'obtient en séparant des expressions par des virgules.

Les deux exemples suivants ciblent les écrans de largeur inférieure ou égale à 768px :

```
<link rel="stylesheet" media="screen and (max-width: 768px)"  
      href="smallscreen.css" type="text/css" />  
  
@media screen and (max-width: 768px) {  
    ...  
}
```

2.3 Les sélecteurs

Les sélecteurs CSS

1. Le sélecteur de type : permet de sélectionner tous les éléments de même type. Par exemple, le code suivant sélectionne tous les `<p>`.

```
p {
    line-height:2em;
}
```

2. Le sélecteur d'ID (#) : permet de sélectionner un élément selon son attribut *id*.

Rappelons que dans une page HTML, on ne peut pas donner le même ID à plusieurs éléments. On opte donc pour des ID lorsqu'on travaille sur un élément unique dans la page : le bloc d'en-tête, le logo de la page ou la barre de navigation.

<pre>#ciel { color:white ; background-color:blue ; } #terre { color:brown; }</pre>	<pre><div id="ciel"> Ce texte est blanc sur un fond bleu. <p id="terre">Ce texte est brun sur un fond bleu.</p> </div></pre>
---	--

3. Le sélecteur de classes (.) : permet de sélectionner des éléments d'une même classe, définis à l'aide de l'attribut *class*. Le principe des classes est similaire à celui des ID mis à part que plusieurs éléments HTML peuvent porter la même classe. On opte donc pour des classes lorsqu'on travaille sur des éléments susceptibles d'être multiples comme des articles, des commentaires, des vignettes, etc.

<pre>.remarque { color:red; } p.detail { color:#CCC; font-size:10px; }</pre>	<pre><div class="remarque"> Ce texte est rouge. </div> <p class="detail">Ce texte est petit et gris.</p> Ce texte ne fait l'objet d'aucune règle CSS. </pre>
---	--

4. Le sélecteur descendant : permet de sélectionner des éléments selon leurs imbrications. Au niveau de la syntaxe, c'est le caractère d'espacement qui exprime la descendance. Par exemple, le code suivant sélectionne tous les `<p>` descendants d'un `<div>`.

<pre>div p { font-weight: bold; }</pre>	<pre><p>Pas gras</p> <div> <p>Gras !</p> <article> <h2>Pas gras</h2> <p>Gras !</p> </article> </div> <div>Pas gras</div></pre>
---	--

Exemple plus complexe combinant les sélecteurs précédents : le sélecteur ci-dessous sélectionne tous les éléments *li* dont la classe est « *info* » contenus dans des éléments *ul* à leur tour contenus dans l'élément *div* dont l'*id* est "menu".

<pre>div#menu ul li.info { color: green; }</pre>	<pre><div id="menu"> <li class="info">Vert ... </div></pre>
--	--

5. Le sélecteur d'enfants (>) : permet de sélectionner un élément directement contenu dans un autre, c'est-à-dire sans élément intermédiaire.

Ce sélecteur concerne les enfants mais pas les autres descendants.

<pre>div>span { color:blue; }</pre>	<pre><div> <p>texte</p> texte en bleu ! <p>texte texte </p> </div></pre>
--	--

6. Le sélecteur universel (*) : représente un élément quelconque. L'exemple suivant revient à sélectionner tous les `` descendants non enfants.

<pre>div * span { color:blue; }</pre>	<pre><div> <p>texte</p> texte <p>texte texte en bleu ! </p> </div></pre>
---	--

7. Le sélecteur de petits frères (~) et de frère adjacent (+) : permet de sélectionner des éléments succédant un élément et ayant le même parent (petits frères) et uniquement le premier de ces petits frères (appelé frère adjacent).

<pre>div~p { color:blue; } div+p { font-weight:bold; }</pre>	<pre><p>texte</p> <div> <p>texte</p> </div> <p>texte en bleu et en gars !</p> <p>texte en bleu !</p> <p>texte en bleu !</p></pre>
--	---

8. Le sélecteur multiple : permet de lister plusieurs sélecteurs séparés par des virgules.

<pre>div, #nav { color : gold ; }</pre>	<p>Le texte de toutes les <code><div></code> et de l'élément dont l'ID est "nav" auront une couleur dorée.</p>
---	--

9. Le sélecteur d'attributs : permet de sélectionner les éléments selon n'importe quel attribut.

Quelques exemples sur l'attribut *lang* :

<pre>/* <a> avec attribut lang */ a[lang] {color:blue;}</pre> <pre>/* <a> avec attribut lang valant "en" */ a[lang='en'] {color:yellow;}</pre> <pre>/* <a> avec attribut lang valant "en" avec suite éventuelle commençant par un tiret */ a[lang ='en'] {color:red;}</pre>	<pre>fr</pre> <pre>en</pre> <pre>en</pre>
---	--

Autres exemples sur l'attribut *class* :

<pre>/* contient « nav » */ p[class*=nav] { color: yellow; }</pre> <pre>/* commence par « nav » */ p[class^=nav] { color: red; }</pre> <pre>/* se termine par « nav » */ p[class\$=nav] { color: green; }</pre>	<pre><p class="bar nav">En jaune</p></pre> <pre><p class="navbar">En rouge</p></pre> <pre><p class="barnav">En vert</p></pre>
---	---

Il est possible d'ajouter un « i » avant le «] » pour demander une insensibilité à la casse ou un « s » pour demander une sensibilité à la casse.

10. Sélecteurs de pseudo-classes : permettent de sélectionner des éléments selon leur état.

:link : les liens non récemment visités
:visited : les liens récemment visités
:hover : les éléments survolés
:focus : les éléments visés (focus)
:active : les éléments cliqués (entre l'appui et le relâchement du bouton)

Remarque :

Lorsque vous définissez l'aspect visuel de vos liens, faites attention à l'ordre de vos sélecteurs de pseudo-classes : un élément « active » est forcément « hover » mais un élément « hover » n'est pas forcément « active » ... Le bon ordre est le suivant :

```
a { text-decoration:none; color:#800; }
a:link { color:#A00; }
a:visited { color:#A06; }
a:hover, a:focus { text-decoration:underline; color:#F00; }
a:active { color:#FF0; }
```



Accessibilité : Le critère WCAG 1.4.1 (A) recommande que les liens au sein d'un texte soient reconnaissables par un autre moyen que la couleur (le soulignement par exemple) sauf si le contraste de couleur entre le texte et le lien est d'au moins 3 et que cet autre moyen de reconnaissance est appliqué au survol et au focus.



Accessibilité : Le critère WCAG 2.4.7 (AA) recommande qu'il soit possible de distinguer visuellement sur quel élément se trouve le focus du clavier (lors du parcours de la page avec la touche TAB).

:first-letter : uniquement la première lettre d'un paragraphe, afin de créer une lettrine.
:first-line : uniquement la première ligne d'un paragraphe.

:after : permet d'insérer un élément de contenu automatiquement après l'élément affecté.
:before : permet d'insérer un élément de contenu automatiquement avant l'élément affecté.

:first-child : cible un élément seulement s'il est le premier élément enfant.
:last-child : cible un élément seulement s'il est le dernier élément enfant.
:only-child : cible un élément seulement s'il est le seul élément enfant.

:lang : cible un élément qui possède l'attribut lang avec une valeur précise.

:required : les champs de formulaire avec attribut *required*

:not() : prend un sélecteur en argument et permet de sélectionner les éléments ne répondant pas à ce sélecteur.

```
a:not(.nav) { ... } /* les liens qui ne sont pas de classe « nav » */
```

Récapitulatif des sélecteurs CSS

Sélecteurs	Syntaxe	Sélection
Sélecteur de type	<code>x { }</code>	Tous les éléments de type <i>x</i>
Sélecteur d'ID	<code>#nom { }</code>	Tous les éléments dont l'ID est <i>nom</i>
Sélecteur d'ID et de type	<code>x#nom { }</code>	Tous les éléments de type <i>x</i> dont l'ID est <i>nom</i>
Sélecteur de classe	<code>.nom { }</code>	Tous les éléments dont la classe est <i>nom</i>
Sélecteur de classe et de type	<code>x.nom { }</code>	Tous les éléments <i>x</i> dont la classe est <i>nom</i>
Sélecteur multiple	<code>x, y { }</code>	Tous les éléments <i>x</i> et tous les éléments <i>y</i>
Sélecteur descendant	<code>x y { }</code>	Tous les éléments <i>y</i> contenus dans un élément <i>x</i> (descendants)
Sélecteur d'enfant	<code>x>y { }</code>	Tous les éléments <i>y</i> directement contenus dans un élément <i>x</i> (= les enfants <i>y</i> de <i>x</i>)
Sélecteur universel	<code>* { }</code>	Tous les éléments
Sélecteur universel	<code>x * y { }</code>	Tous les éléments <i>y</i> contenus dans des éléments quelconques, eux-mêmes contenus dans des éléments <i>x</i> (= les descendants <i>y</i> non enfants de <i>x</i>)
Sélecteur de frères adjacents	<code>x+y { }</code>	L'élément <i>y</i> s'ouvrant juste après la fermeture d'un élément <i>x</i> et ayant le même parent
Sélecteur de petits frères	<code>x~y { }</code>	Tous les éléments <i>y</i> s'ouvrant après la fermeture de <i>x</i> et ayant le même parent
Sélecteur de pseudo-classes	<code>x:z { }</code>	Tous les éléments <i>x</i> dans l'état <i>z</i> .

Exercice : les sélecteurs CSS

Suite aux règles CSS précisées ci-dessous, de quelles couleurs s'afficheront les mois « Septembre », « Octobre », « Novembre », « Décembre » et « Janvier » ? Lesquels seront en gras ?

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
    <title>Exercice CSS sur les sélecteurs</title>
    <style>
        div { color: yellow; }
        div p { color: red; }
        div span { font-weight: bold; color: black; }
        div > span { color: blue; }
        div * span { color: green; }
    </style>
</head>
<body>
    <div>
        <p> Septembre </p>
        <p> Octobre
            <span> Novembre </span> </p>
        <span> Décembre </span>
        Janvier
    </div>
</body>
</html>
```

Exercice : rédigez des règles CSS

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
    <title>Exercice CSS sur les sélecteurs</title>
    <style>
        </style>
</head>
<body>
    <header id="entete">
        <h1>Titre vert et gras</h1>
    </header>
    <div>
        <h2>Sous-titre bleu et gras</h2>
        <p>Texte bleu</p>
        <p class="remarque">Texte rouge</p>
        <p>Texte bleu et <span>texte gris</span></p>
    </div>
</body>
</html>
```

2.4 Les déclarations et leurs priorités

Déclarations, héritage et cascade

Les déclarations CSS permettent de modifier l'aspect des éléments et sont toujours de la forme ***propriété:valeur;***

Certaines déclarations, principalement celles portant sur les propriétés de mise en forme des textes (*color*, *font-style*, *font-family*, *text-decoration*, etc.), sont héritées par les éléments descendants de l'élément sélectionné. Cet héritage peut toutefois être contredit par une nouvelle déclaration sur l'un des descendants. Exemple :

```
<html>
<head>
<style>
  body {
    color:green;
    font-weight:bold;
    margin:5%;
  }
  div {
    color:turquoise;
  }
</style>
</head>

<body>
  <div>
    <p>Texte</p>
  </div>
</body>
</html>
```

Dans cet exemple, 3 déclarations portent sur `<body>` : une couleur de texte verte, une graisse (*bold*) et une marge de 5%.

La marge, comme la plupart des propriétés CSS, n'est pas héritée par les descendants et ne s'applique donc qu'à l'élément visé : `<body>`.

La couleur verte et la graisse du texte, sont héritées par tous les descendants de `<body>`. Cependant, la couleur turquoise appliquée au `<div>`, contredit la couleur verte du `<body>` et est héritée par tous ses propres descendants.

En conclusion, le texte de `<p>` sera turquoise et gras !

C'est le principe de cascade CSS.

Cette faculté d'héritage en cascade nous évite de coder les propriétés de mise en forme des textes pour chaque descendant.

Les règles CSS du navigateur

Certains éléments disposent de leurs propres déclarations CSS par défaut. Par exemple, les liens sont bleus et soulignés ; les titres et les paragraphes possèdent des marges ; les éléments des listes possèdent des puces ; etc.

Ces règles CSS sont de la forme `a { color:blue; text-decoration:underline; }` et sont à prendre en compte comme toute autre règle CSS lors de l'héritage en cascade : pour changer la couleur des liens, cibler un ancêtre ne suffit pas, il faut cibler les liens `<a>`.

Imposer une déclaration prioritaire

Il est possible de rendre une déclaration prioritaire sur d'autres déclarations concernant un même élément.

```
body { background-color: red !important; }
```

Toutefois, il est conseillé de ne pas abuser de cette syntaxe et de la conserver en cas de dernier recours.

Priorité des déclarations

Un même élément peut être concerné par plusieurs règles CSS. Si les propriétés appliquées sont différentes, elles se combinent : du bleu et du gras par exemple. Mais si la même propriété est appliquée par plusieurs règles, il y a conflit... et une seule propriété l'emporte.

Pour connaître la **priorité** d'une déclaration, une formule assez simple permet de donner une valeur à chaque déclaration : la valeur la plus haute l'emporte ; en cas d'égalité, la dernière déclaration dans le code l'emporte.

Comment calculer la priorité des déclarations CSS ? En décomposant le sélecteur et en additionnant les valeurs suivantes :

1. La commande ***!important*** vaut 1000.
2. Les déclarations écrites sous forme **d'attribut style** valent 1000 aussi mais passent après la commande ***!important***...
3. Chaque **sélecteur d'ID** vaut 100.
4. Chaque **sélecteur de classe** ou de pseudo-classe vaut 10.
5. Chaque **sélecteur de type** ou de pseudo-élément vaut 1.
6. Le sélecteur universel « * » vaut 0.

Remarque : dans ce calcul, il n'y a pas de report de rang, c'est-à-dire que 11 sélecteurs de classe ne valent pas plus qu'un sélecteur d'ID.

Exemple de règle CSS :

```
#header nav#navtop a {  
    color: red !important;  
    font-size: 1.25rem;  
}
```

Dans cet exemple, la couleur rouge a une priorité de 1202 et la taille du texte a une priorité de 202.

Pour une meilleure lisibilité du document CSS, on commence en général par coder les règles sur les éléments ancêtres et on termine par les éléments plus précis, leurs descendants.

Exemple pour mieux comprendre les priorités des déclarations

```
<html>
<head>
<title>Exemple 1</title>
<style>

div { font-weight:bold; }
div { color:red; }
p { color:green !important; }
#txt { color:yellow; }
div>p { color:orange; }

</style>
</head>

<body>
<div>
  <p id="txt">Vert et gras</p>
</div>
</body>
</html>
```

Dans cet exemple, le `<p>` est concerné par les toutes les règles CSS !

La 1^{re} règle appliquant du gras s'applique au texte car aucune autre règle ne travaille la propriété `font-weight`.

La déclaration appliquant du rouge n'a aucune chance contre les déclarations suivantes car elle porte sur un élément ancêtre du `<p>` alors que les déclarations suivantes portent sur le `<p>`.

La déclaration "green" a une priorité de 1001.
La déclaration "yellow" a une priorité de 100.
La déclaration "orange" a une priorité de 2.

Le vert l'emporte.

Exercice : priorité des sélecteurs

De quelle couleur seront affichés les noms suivants ?

Harry	
Hermione	
Ron	
Drago	
Hagrid	
Hedwige	

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
    <title>Exercice CSS sur les sélecteurs</title>
    <style>
        body { color:white; background-color:#444; }
        h1, h2, h3 { color:gold; }
        h2+p { color:pink; }
        .movie small { color:grey; }
        .movie .content p { color:violet; }
        .movie p+p { color:brown; }
        .movie footer * { color:orange; }
        #content>p { color:red; }
    </style>
</head>
<body>
    <main id="content">
        <article class="movie">
            <header>
                <h2>Harry</h2>
            </header>
            <div class="content">
                <p>Hermione <small>et Ron</small></p>
                <p>Drago</p>
            </div>
            <footer>
                <h3>Hagrid</h3>
                Hedwige
            </footer>
        </article>
    </main>
</body>
</html>
```



Exercice : priorité des sélecteurs

Voici un code CSS et HTML. De quelle couleur seront affichés les noms suivants ?

Naruto		Hinata	
Sakura		Kiba	
Ino		Shino	
Shikamaru		Neji	
Choji		Rock Lee	

```
<!DOCTYPE html>
<html>
<head>
    <title> Exercice CSS sur les sélecteurs </title>
    <style>
        body { color:black; }
        h1 div { color:gold; }
        p+div { color:turquoise; }
        p > span { color:blue; }
        strong { color:green; }
        .manga strong { color:orange; }
        strong.manga { color:red; }
        li li { color:yellow; }
        ul { color:silver; }
    </style>
</head>
<body>
<div>
    <h1> Naruto </h1>
    <p style="color:violet;"> Sakura
        <span> Ino </span>
    </p>
    <div> Shikamaru
        <h1> <span> Choji </span> </h1>
        <strong class="manga"> Hinata </strong>
        <ul>
            <li class="manga"> <strong> Kiba </strong> </li>
            <li>
                <ul>
                    <li> Shino </li>
                    <li class="manga"> Neji </li>
                </ul>
            </li>
            <li> <strong id="manga"> Rock Lee </strong> </li>
        </ul>
    </div>
</div>
</body>
</html>
```

2.5 Les unités de mesure

Remarque : attention à ne pas séparer la valeur et l'unité d'un espace !

Attention à toujours préciser l'unité à côté de la valeur. C'est obligatoire à part si la valeur est zéro ou dans le cas de quelques propriétés comme l'interligne.

Unités de niveau 4

Les unités de niveau 4 (vi, vb, ic, lh, rlh, etc.) sont en développement. Il est trop tôt pour les utiliser.

Les dimensions

Pixel : **320px**

Pourcentage : **100%**

- se réfère à la hauteur (*height*) du parent si utilisé pour une propriété « verticale » : *height*, *min-height*, *top* ou *bottom*.
- se réfère à la largeur (*width*) du parent dans les autres cas : *width*, *min-width*, *margin*, *padding*, *left*, *right*, *text-indent*, etc.
- se réfère au *font-size* du parent si utilisé pour : *font-size* ou *line-height*
- n'est pas permis dans certaines propriétés : *border-width*, *letter-spacing*, etc.

Vue (vw, vh, vmin, vmax) : Des unités dépendant de la vue (taille de la fenêtre) :

- **1vw** vaut 1/100 de la largeur de la fenêtre
- **1vh** vaut 1/100 de sa hauteur
- **1vmin** vaut le plus petit des 2
- **1vmax** le plus grand des 2

Les dimensions pour l'impression

Centimètre : **2cm**

Millimètre : **18mm**

Pouce : **1in** (1 inche = 2.54 cm)

Pica : **1pc** (1 pica = 12pt)

Point : **12pt** (1 point = 1/72 in ; une police de 12pt équivaut à 16px)

Les cadratins

Cadratin relatif à la largeur de la lettre « m » : **1em**

Cadratin relatif à la hauteur de la lettre « x » : **1ex** (peu utilisé)

Appliqué à la propriété *font-size*, le cadratin *em* agit comme un multiplicateur du *font-size* de l'élément parent. Appliqué à une autre propriété, il multiplie le *font-size* de l'élément lui-même.

```
html { font-size:16px; }
body { font-size:1.5em; } /* font-size=24px */
h1 { font-size:2em; margin:1em; } /* font-size=48px margin:48px */
```

Root em (rem) : identique à *em*, le cadratin *rem* se base sur le *font-size* de la racine. Exemple :

```
html { font-size:16px; }
body { font-size:1.5rem; } /* font-size=24px */
h1 { font-size:2rem; margin:1rem; } /* font-size=32px margin=16px */
```

Les angles

Degrés : **90deg** Grades : **-0.5grad**
($180\text{deg} = 200\text{grad} = 0.5\text{turn} = 3.1416\text{rad}$)

Radian : **1rad**

Tour : **10turn**

Le temps et les fréquences

Seconde : **3s**
Milliseconde : **400ms**

Hertz : **10000Hz**
KiloHertz : **10kHz**

calc()

Introduite en CSS3, la fonction CSS *calc()* permet un calcul utilisant les opérateurs + - * / et portant sur des valeurs pouvant présenter des unités différentes.

Exemple :

```
width : calc(50% - 16px) ;
```

Attention, les espaces autour de l'opérateur ont leur importance.

Variables CSS

```
:root {  
    --background:#333;  
    --text:#ddd;  
}  
  
body {  
    background-color:var(--background);  
    color:var(--text);  
}
```

Très pratiques pour s'assurer de la cohérence graphique d'un document, les variables CSS permettent de centraliser les valeurs récurrentes dans la feuille de style.

2.6 Les propriétés

Couleur

Le CSS permet de modifier la couleur du texte (***color***), du fond (***background-color***), des bordures (***border-color***), la couleur d'accentuation des champs de formulaire (***accent-color***), etc.

```
color: #D33 ;
```

Pour coder des couleurs, le CSS supporte de nombreuses syntaxes différentes :

- **#rrggbb**, où *rr*, *gg* et *bb* sont des nombres hexadécimaux à deux chiffres, de 00 à FF, pour le rouge, le vert et le bleu.
- **#rgb**, où *r*, *g* et *b* sont des nombres hexadécimaux à un chiffre, de 0 à F, pour le rouge, le vert et le bleu. Cette écriture est une abréviation de la précédente avec un choix de couleur plus restreint puisque #FOB revient à #FF00BB.
- **#rrggbbaa**, où *aa* est la transparence (00 pour transparent, FF pour opaque), déconseillé à cause de sa faible compatibilité.
- **#rgba**, où *a* est la transparence (0 pour transparent, F pour opaque), déconseillé à cause de sa faible compatibilité.
- **rgb(r,g,b)**, où *r*, *g* et *b* sont des valeurs entières de 0 à 255, pour le rouge, le vert et le bleu.
- **rgb(r,g,b)**, où *r*, *g* et *b* sont des pourcentages, pour le rouge, le vert et le bleu.
- **rgba(r,g,b,a)**, où *a* est le niveau de transparence. Par exemple 0.75 pour 75%.
- **hsl(h,s,l)**, où *h* (*hue*) est la teinte, *s* la saturation et *l* (*lightness*) la clareté.
- **hsla(h,s,l,a)**, où *a* est le niveau de transparence.
- par un nom en anglais. À savoir que le choix est alors plus restreint et que certains noms de couleurs ne sont pas connus de tous les navigateurs.
- Par certains mots-clés : ***transparent***, ***currentcolor*** (valeur de *color*), ***inherit***, ***initial***, ***unset***.



Accessibilité : Le critère WCAG 1.4.3 (AA) recommande un contraste d'au moins 4,5 entre la couleur d'un texte et la couleur de fond ; un contraste d'au moins 3 si le texte est de grande taille (24px ou 18.66px gras).

Besoin d'un outil pour calculer le contraste ? <https://contrast-ratio.com/>

Opacité

Attention à ne pas confondre l'opacité et la transparence (*rgba*, *hsla*). L'opacité s'applique à tous les descendants d'un élément et attend une valeur réelle entre 0 et 1. Il s'agit d'une propriété **CSS3**.

```
opacity: 0.75 ;
```

Visibilité

La propriété ***visibility*** permet de rendre un élément invisible tout en conservant son emplacement. Cette propriété peut prendre les valeurs ***visible*** ou ***hidden***.

```
visibility: hidden ;
```

Police de caractères

La police de caractères, souvent appelée « fonte » malgré une nuance : une police de caractères est un ensemble de fontes, et une fonte est une combinaison de police, de taille et de graisse (niveau de gras).

Comme toutes les machines ne sont pas équipées de toutes les polices, la propriété ***font-family*** autorise de lister plusieurs noms de police, ordonnés et séparés par des virgules. Le navigateur tente la première, si elle n'est pas disponible, il tente la suivante, etc.

Il est d'usage de proposer plusieurs polices (semblables) et de terminer par un générique (*serif*, *sans-serif*, *monospace*, *cursive* ou *fantasy*) : les valeurs ***serif*** et ***sans-serif*** représentent les familles de polices avec et sans empattements, alors que la valeur ***monospace*** représente la famille de polices mono-espacées (chaque caractère a la même largeur), ***cursive*** la famille de polices simulant l'écriture à la main et ***fantasy*** la famille de polices décoratives.

`font-family: Trebuchet MS, Arial, sans-serif;`

Les polices les plus fréquentes sur le Web sont actuellement : Arial, Helvetica Neue, Verdana, Open Sans, Helvetica, Georgia, Tahoma, Menlo, Trebuchet MS, Monaco, Roboto, Lato, Lucida Grande, Consolas, Courier New, Oswald, Times New Roman, etc.

Certaines polices sont en réalité des sources d'icônes : Font Awesome, Glyphicons Halflings, Genericons, etc.

Si l'envie vous prend d'utiliser une **police rare ou exotique**, n'oubliez pas que cette police a de faibles chances d'être installée sur les machines de vos visiteurs et vous devez donc l'incorporer à votre site. Prévoyez un dossier « fonts » pour disposer les fichiers *woff2*, *woff* ou *ttf* et utilisez le code suivant, au début de votre fichier CSS :

```
@font-face {
    font-family: "Starcraft";
    src: url('fonts/Starcraft Normal.woff2'),
         url('fonts/Starcraft Normal.woff'), ;
}
@font-face {
    font-family: "Starcraft";
    font-weight: bold;
    src: url('fonts/Starcraft Bold.woff2'),
         url('fonts/Starcraft Bold.woff'), ;
}
```

La police nommée « Starcraft » peut alors être utilisée de cette manière :

`h1, h2, h3 { font-family: Starcraft, Arial, sans-serif; }`

Il existe de nombreux formats de fichiers de polices. Le format « *woff2* » assure une bonne compatibilité (>95%). Ajouter d'autres formats améliore légèrement la compatibilité.

Pour information, Alsacréations partage un projet Github "Webfonts" où il vous est possible de récupérer et héberger plusieurs dizaines de familles de polices optimisées pour le Web : <https://github.com/alsacreations/webfonts>.

Enfin, deux solutions s'offrent à vous : placer les fichiers sur votre serveur ou profiter de services en ligne de type CDN comme Google Font ou Typekit. Malgré quelques avantages apportés par les CDN, il est conseillé de placer les fichiers sur votre serveur.

Mise en forme des textes

La première chose qu'on veut apprendre à changer, c'est souvent l'apparence du texte : la police, la couleur, la taille, etc. L'utilisation du CSS fait complètement disparaître l'usage de l'antique élément HTML ``. Voici l'essentiel des déclarations de police :

1. **color** : la couleur du texte
`color: #333 ;`

2. **font-family** : la police de caractères (voir page précédente pour plus d'informations).
`font-family: Trebuchet MS, Arial, sans-serif;`

3. **font-size** : la taille de la police. On peut lui donner une valeur descriptive (***xx-small, x-small, small, medium, large, x-large, xx-large***), une valeur relative à la taille actuelle (***larger, smaller*** ou un pourcentage), ou une taille fixée.

`font-size: 1.5rem;`

Les navigateurs appliquent généralement une taille par défaut de 16px (équivalent de 12pt). Mais certaines personnes malvoyantes configurent leur navigateur sur une `font-size` plus grande. Pour ne pas contredire ce paramètre, il est de bon usage d'éviter les valeurs exprimées en `px`, et de s'en tenir aux unités proportionnelles : en % pour l'élément racine, en `em` ou `rem` pour le reste.

4. **font-style** : ***italic, oblique*** ou ***normal***. Souvent, l'italique a ses propres caractères alors que l'oblique n'est qu'une version inclinée des caractères de base.

`font-style: italic;`

5. **font-weight** : ***bold*** pour gras, ***normal*** pour non-gras, ***bolder*** et ***lighter*** pour plus ou moins gras que l'élément parent. Il est possible aussi de préciser une valeur numérique entre 100 et 900 par pas de 100, correspondant à différents niveaux de gras (si toutefois la police le supporte).

`font-weight: bold;`

6. **line-height** permet de changer la taille de l'interligne. La valeur par défaut est ***normal***. Les cadratins (em) et les pourcentages sont préférables aux pixels (px). Cette propriété accepte les valeurs sans unité, la valeur agit alors comme un multiplicateur de la taille d'interligne d'origine.

`line-height: 1.5;`

7. **text-align** : l'alignement du texte, peut prendre les valeurs : ***left, right, center*** et ***justify***. Attention, cette propriété est destinée à être appliquée à des conteneurs (blocs) et pas aux éléments en-lignes.

`text-align: center;`

8. **text-decoration** : permet de modifier ou de supprimer la décoration du texte. Par exemple, si vous souhaitez ne pas souligner vos liens, vous pouvez donner à cette propriété la valeur **none**. A l'opposé, il est possible de souligner avec **underline** ou surligner avec **overline** ou barrer avec **line-through**.

```
text-decoration: none;
```

Cette propriété compte 3 sous-propriétés (souffrant de problèmes de compatibilité) : **text-decoration-line** (*none, underline, overline, line-through*), **text-decoration-color** (une couleur) et **text-decoration-style** (*solid, wavy, double, dotted, dashed*). Les 3 valeurs peuvent être appliquées en une seule fois :

```
text-decoration: wavy purple underline ;
```

9. **text-shadow** : permet d'appliquer une ombre au texte, ou plusieurs ombres en les séparant par des virgules. La propriété **box-shadow** est en tous points semblable mais s'applique aux éléments blocs. Chaque ombre comporte 4 valeurs : le décallage en x, le décallage en y, le rayon et la couleur.

```
text-shadow: 1px 1px 2px #999;
```

10. **vertical-align** : permet de modifier l'alignement vertical d'un élément en-ligne par rapport à d'autres éléments situés sur la même ligne. Par exemple l'alignement vertical d'une image par rapport à un texte ou à d'autres images.

Peut prendre les valeurs **sub** (indice), **super** (exposant), **baseline** (sur la ligne), **top**, **middle**, **bottom**, **text-top**, **text-bottom**. Des valeurs numériques sont également possibles pour décaler verticalement des portions de texte.

```
vertical-align: middle;
```

Mise en forme des textes – propriétés peu utilisées



11. **font-variant** : *small-caps* pour les petites capitales, ou *normal*.
font-variant: small-caps;

12. **font-stretch** : permet de distendre le texte. Les valeurs possibles sont : *ultra-condensed*, *extra-condensed*, *condensed*, *semi-condensed*, *normal*, *semi-expanded*, *expanded*, *extra-expanded* et *ultra-expanded*.
font-stretch: condensed;

13. **hyphens** : indique au navigateur comment gérer les traits d'union lors des sauts de ligne. La propriété **hyphens** peut prendre les valeurs *none*, *manual* ou *auto*.
hyphens: none;

14. **letter-spacing** : permet de modifier l'espace entre les lettres.
letter-spacing: 0.2em;

15. **text-direction** : permet de modifier le sens d'écriture d'un texte (*ltr* « left-to-right » ou *rtl*).
text-direction: rtl;

16. **text-indent** : permet de modifier l'indentation de la première ligne d'un texte.
text-indent: 1em;

17. **text-justify** : lorsque le texte est justifié, permet de modifier la méthode de justification.
 Valeurs possibles : *auto*, *inter-word*, *inter-character*, *none*.
text-justify: inter-character;

18. **text-transform** : permet de modifier la casse du texte. Les valeurs possibles sont *none* (par défaut), *lowercase* (aucune majuscule), *uppercase* (que des majuscules) ou *capitalize* (chaque mot commence par une majuscule).
text-transform: lowercase;

19. **user-select** : permet d'empêcher l'utilisateur de sélectionner le texte.
user-select: none;

20. **word-spacing** : permet de modifier l'espace entre les mots
word-spacing: 1em;

Curseur	I auto	↔ move	↗ no-drop	↔ col-resize
	↔ all-scroll	pointer	🚫 not-allowed	⤵ row-resize
La propriété <i>cursor</i>	+ crosshair	⏳ progress	↔ e-resize	⤲ ne-resize
permet de modifier le curseur de la souris lorsque celui-ci survole un élément.	"default	I text	↑ n-resize	⤳ nw-resize
	?	➡ vertical-text	↓ s-resize	⤴ se-resize
	I inherit	⌚ wait	↔ w-resize	⤵ sw-resize

cursor: pointer;

Dans les champs de formulaire, la couleur du curseur *text* peut être personnalisée avec la propriété **caret-color**.

Colonnes multiples

Il est ais  en CSS de disposer un texte sur plusieurs colonnes. Appliqu es   un conteneur, les d clarations ci-dessous vont provoquer une disposition en 3 colonnes avec des goutti res de 1em et des r gles (barres de s paration) de 1px gris-clair continu.

```
column-count: 3;  
column-gap: 1em;  
column-rule-style: solid;  
column-rule-width: 1px;  
column-rule-color: lightgrey;
```

Lorem ipsum dolor sit amet, consectetur adipisciing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper

suscipit lobortis nisl ut aliquip
ex ea commodo consequat.
Duis autem vel eum iriure
dolor in hendrerit in vulputate
velit esse molestie consequat,
vel illum dolore eu feugiat
nulla facilisis at vero eros et
accumsan et iusto odio

dignissim qui blandit praesent
luptatum zzril delenit augue
duis dolore te feugait nulla
facilisi. Nam liber tempor
cum soluta nobis eleifend
option congue nihil imperdiet
doming id quod mazim
placerat facer possim assum.

Certains éléments peuvent empiéter sur plusieurs colonnes :

```
h3 { column-span: 2; }
```

Largeur et Hauteur

La largeur du contenu d'un élément se définit avec la propriété ***width*** et sa hauteur avec ***height***. La valeur ***auto*** demande la taille par défaut.

```
width:80%;  
height:220px;
```

Des valeurs minimales et maximales peuvent compléter ces 2 propriétés :

```
min-width:100px;  
max-width:960px;
```

Tout élément bloc se compose d'un contenu enrobé de trois zones : le remplissage (*padding*), la bordure (*border*) et la marge (*margin*). Les propriétés *width* et *height* n'agissent que sur le contenu.

En définissant une largeur d'élément de par exemple 400px, vous dimensionnez le contenu de l'élément. La place occupée par l'élément peut être plus élevée s'il possède des marges ou des bordures : 420px dans cet exemple.

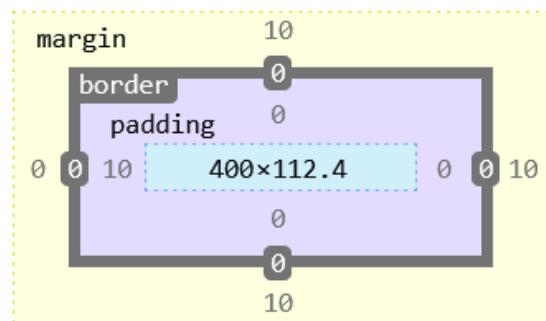


Figure 4 : inspection d'élément avec Firefox

Bordures et marges

1. Marges (*margin*) - Dans le cas des marges, seule leur étendue peut être définie. Pour cela, on utilise les propriétés ***margin-top***, ***margin-right***, ***margin-bottom*** et ***margin-left***. On peut spécifier l'étendue des quatre marges en une fois avec ***margin***. Voici différentes syntaxes permises :

```
margin-top : 120px ; /* marge supérieure de 120px */
margin : 0 ; /* 4 marges nulles */
margin : 80px auto ; /*verticales de 80px ; horizontales automatiques*/
margin : 10px 0 20px 2em ; /* 4 valeurs : top, right, bottom, left */
```

2. Remplissage (*padding*) - Les marges intérieures (***padding***) se règlent exactement comme les marges extérieures (***margin***), à ceci près que la propriété se nomme ***padding*** au lieu de ***margin*** et que ***padding*** n'accepte que des valeurs positives.

Si vous appliquez un *bakcground* à un élément, celui-ci s'étend sur le contenu et le *padding*.

3. Bordures (*border*) – La bordure se situe entre le remplissage et la marge. La bordure possède trois caractéristiques : ***width*** (l'épaisseur), ***color*** et ***style***. Comme pour les marges et le remplissage, la bordure peut être définie différemment pour les 4 côtés.

Les valeurs possibles pour ***border-style*** sont ***none*** (aucun, par défaut), ***hidden*** (invisible), ***dotted*** (pointillé), ***dashed*** (tirets), ***solid*** (continu), ***double*** (épais), ***groove*** (*relief*), ***ridge*** (bourrelet), ***inset*** (creux) et ***outset*** (relief). Voici différentes syntaxes :

```
border : 2px #333 solid ; /*une bordure continue anthracite de 2px*/
border-width : 5px 10px ; /*épaisseurs horizontale et verticale*/
border-top : 3px dotted #f00 ; /*une bordure top rouge
pointillée*/
border-bottom-color : red ; /*une couleur rouge en bas*/
```

Depuis le **CSS3**, il est possible d'arrondir les coins des éléments (qu'ils disposent ou non d'une bordure). La propriété ***border-radius*** prend comme valeur le rayon de l'arrondi en question.

```
border-radius : 8px ; /*hop, 4 coins arrondis*/
border-radius : 8px 0 ; /*8px d'arrondi top-left et bottom-right*/
border-radius : 8px 0 4px 0 ; /*4 arrondis différents (en partant de
top-left et en tournant jusque bottom-left)*/
border-radius : 70% 40% 30% 60% / 60% 70% 40% 60% ; /*arrondi heu... */
```

Il est possible de charger une image de bordure avec ***border-image***, puis de la régler avec ***border-image-outset***, ***-repeat***, ***-slice***, ***-source*** et ***-width***.

Les éléments **<table>** supportent 2 propriétés supplémentaires :

- ***border-collapse*** permet de définir le comportement de 2 bordures jointes : ***separate*** (côte à côte, valeur par défaut) ou ***collapse*** (fusionnées).
- ***border-spacing*** permet de définir la distance entre 2 bordures.

4. Contour (*outline*) – Moins utilisée que la bordure, le contour est une seconde bordure à l'extérieur de la première, mais sans consistance et sans possibilité d'arrondi.

```
outline : 2px red solid ; /*utilisation similaire à border*/
outline-offset : 4px ; /* distance entre le contour et l'élément */
```

Calcul des dimensions des blocs

La propriété *width* détermine la largeur du contenu d'un bloc, c'est-à-dire la zone disponible pour le texte, etc. Si l'on définit une marge intérieure *padding*, celle-ci augmente la largeur totale de l'élément. Mais il est possible de changer ce comportement de façon à ce que le *padding* se soustrait au contenu avec *box-sizing:border-box*.

Ce bloc a une largeur apparente de 140px dont 100px pour disposer le contenu :

```
div { box-sizing:content-box; width:100px; padding:20px; }
```

Ce bloc a une largeur apparente de 100px dont 60px pour disposer le contenu :

```
div { box-sizing:border-box; width:100px; padding:20px; }
```

Cette propriété s'applique en largeur et en hauteur.

Alignment : centrer un élément

Centrer un élément dépend de sa nature : les contenus de type en-ligne (éléments HTML en-ligne et textes) se centrent avec *text-align:center*; alors qu'un élément de type bloc se centre avec *margin:auto* ;

Si l'on souhaite centrer un **contenu en-ligne** (un texte, un ** ou un ** par exemple), il faut utiliser la propriété ***text-align***. Mais attention, cette propriété ne doit pas être appliquée à ce contenu mais à son conteneur !

Cette propriété ***text-align*** décrit donc l'alignement des contenus en-ligne à l'intérieur du conteneur ciblé, et peut prendre plusieurs valeurs : ***left*** (par défaut), ***center***, ***right*** ou ***justify***. Cette dernière valeur est toutefois à éviter si la place disponible est étroite pour éviter des espaces démesurés entre les mots.

Soit le code HTML suivant : ***<div> </div>***

Voici comment centrer l'image : ***div { text-align:center; }***

Si l'on souhaite centrer un **élément bloc**, il faut lui appliquer une marge extérieure horizontale de valeur ***auto*** (***margin:0 auto;***). Les marges gauche et droite prennent alors des valeurs, ce qui a pour effet de centrer le bloc horizontalement. Attention, il faut évidemment aussi que cet élément n'ait pas la même largeur que son parent car dans ce cas, le centrer n'aura aucun effet...

Remarquez que cette méthode ne permet pas de centrer un bloc verticalement.

Soit le code HTML suivant : ***<div id="main">...</div>***

Voici comment centrer le bloc : ***#main { margin:0 auto; width:960px; }***

Listes à puces

Le style des listes, dont dépend la puce, que nous avons vu en HTML est sensé être spécifié en CSS. La propriété ***list-style*** comporte 3 caractéristiques : ***image*** (image de la puce), ***position*** (de la puce) et ***type*** (admettant comme valeurs ***disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none***). Comme pour les autres propriétés comportant des caractéristiques, il est possible de les combiner.

```
list-style: none ;
```

Arrière-plan

Les principales propriétés permettant de modifier l'arrière-plan sont :

- **background-color** : couleur d'arrière-plan.
- **background-image** : image de fond ou dégradé.
- **background-repeat** : *repeat* pour répéter l'image, *repeat-x* pour la répéter horizontalement, *repeat-y* pour la répéter verticalement et *no-repeat* pour ne pas la répéter.
- **background-position** : deux valeurs séparées d'un espace pour indiquer la position horizontale puis verticale. La composante verticale admet les valeurs **top**, **bottom** et **center** alors que l'horizontale admet les valeurs **right**, **left** et **center**, mais les valeurs peuvent également être spécifiées en pourcents, en pixels, etc.
- **background-attachment** : *scroll* (par défaut) ou *fixed* selon que l'image de fond défile avec le reste de la page ou pas.
- **background-size**: **x y**; taille horizontale et verticale de l'image de fond. Peut également prendre la valeur **cover** pour recouvrir toute la surface ou **contain** pour afficher l'image en entier.

Exemple d'une image « image.png » qui s'affiche sur un fond rouge, qui ne se répète pas et qui est située dans le haut de la page à 25% de la largeur de la page (côté gauche) :

```
background-color: red;
background-image: url(image.png);
background-repeat: no-repeat;
background-position: 25% 0%;
background-attachment: fixed;
background-size: cover;
```

Le tout peut être résumé en une seule ligne (peu importe l'ordre des valeurs), à part le *background-size* paru plus tard qui ne se combine pas avec le reste :

```
background: red url(image.gif) no-repeat fixed 25% 0%;
background-size: cover;
```

Arrière-plan – dégradés



background-image: linear-gradient(purple, blue) ;

La propriété *background-image* peut servir à appliquer un dégradé de couleur. Ci-dessus, un exemple simple de dégradé linéaire.

background-image: linear-gradient(to right, blue 0%, #0ff 50%, #0f0 75%) ;

Ci-dessus, un exemple complexe de dégradé linéaire dirigé vers la droite (*to right*) de 3 couleurs successives en spécifiant le début de chaque couleur (le bleu à 0%, le #0ff à 50%, etc.). La valeur *to right* peut être remplacée par *to bottom*, *to left*, *to top*, *to bottom-left*, etc. mais aussi par un angle exprimé en degrés, en radians ou en radians : *to 60deg*

background-image: radial-gradient(circle at center, white, green);

Le dégradé radial est également disponible. La valeur *circle* peut être remplacée par *ellipse*.

Les possibilités de dégradés sont plus étendues que celles proposées dans ce cours. N'hésitez pas à faire un tour sur le Web pour découvrir d'autres possibilités de dégradés.

Arrière-plan – propriétés peu utilisées



Couper l'image de fond :

background-clip: ... ;

Peut prendre les valeurs **border-box**, **padding-box** ou **content-box** pour limiter l'arrière-plan à la bordure, à la marge intérieure ou au contenu.

Origine de l'image de fond :

background-origin: ... ;

Peut prendre les valeurs **border-box**, **padding-box** (par défaut) ou **content-box** pour déterminer le point d'origine de l'image de fond.

Images de fond multiples :

background-image: url(...), url(...) ;

background-position: x1 y1, x2 y2 ;

Lorsque plusieurs images sont spécifiées, les autres propriétés prennent, soit plusieurs valeurs (en respectant l'ordre), soit une valeur unique qui est alors commune à toutes les images.

Exemple d'écriture condensée pour images de fond multiples :

```
background: url(image1.png) no-repeat top left,
            url(image2.png) no-repeat top right,
            url(image3.png) repeat-x bottom left;
```

Guillemets



Il est possible de personnaliser les guillemets provoqués par l'élément <q> :

```
q { quotes: "«" "»" "⟨" "⟩"; }
```

La propriété **quotes** prend 4 valeurs séparées par des espaces, chacune entre guillemets. Les 2 derniers symboles ne sont utilisés que si la citation contient une autre citation.

Voici différents symboles de guillemets, dont l'utilisation varie selon la langue :

" " ' ' < > << >> ' ' " " "

La graphie française est la suivante « citation ».

Saut de page (pour le *print*)



Uniquement appliquée en cas d'impression de la page web, les propriétés **page-break-before**, **page-break-inside** et **page-break-after** permettent de gérer les sauts de page.

```
page-break-before: auto;
page-break-inside: avoid;
page-break-after: always;
```

Les valeurs possibles sont :

auto : gestion automatique (par défaut)

always : provoque toujours un saut de page

avoid : évite de provoquer un saut de page si possible

left : provoque un saut de page et la prochaine page sera une page gauche

right : provoque un saut de page et la prochaine page sera une page droite

Ce qui dépasse

overflow:hidden;

La propriété **overflow** permet de gérer les éléments qui dépassent de leur cadre en prenant les valeurs suivantes :

- **auto** : augmente la taille des conteneurs si le contenu dépasse
- **hidden** : rogne tout ce qui dépasse
- **scroll** : barre de défilement
- **visible** : ce qui dépassé est visible

Les propriétés **overflow-x** et **overflow-y** permettent de gérer séparément ce qui dépasse horizontalement ou verticalement.

Rognures

La propriété CSS3 **clip-path** permet d'empêcher une partie d'un élément de s'afficher. La région rognée peut faire appel à un SVG externe ou à une instruction SVG *path* ou à une fonction de forme :

```
clip-path: circle(50%);  
clip-path: circle(25% at 25% 25%); /*centre du cercle déplacé*/  
clip-path: ellipse(50% 30%);  
clip-path: polygon(50% 0, 100% 50%, 50% 100%, 0 50%); /*losange*/  
clip-path: path('M 0 200 L 0,75 A 5,5 0,0,1 150,75 L 200 200 z');
```

Un générateur de *clip-path* en ligne : <https://bennettfeely.com/clippy/>

Transitions

transition-property : Définit les propriétés CSS qui subiront une transition. Par exemple, les marges, la largeur ou la couleur.

transition-duration : Définit la durée de la transition.

transition-timing-function : Définit la vitesse (facultatif) et peut prendre les valeurs *linear*, *ease*, *ease-in*, *ease-out*, *ease-in-out*.

transition-delay : Définit l'avance ou le retard que prendra la transition vis à vis de son déclenchement (facultatif).

Exemple :

```
a {  
    color: #f00;  
    transition-property: color;  
    transition-duration: 1s;  
}  
a:hover, a :focus {  
    color: #f0f;  
}
```

Il est possible de spécifier les 4 propriétés avec la seule propriété *transition*. Ou encore d'utiliser *transition* pour le préciser que la durée d'une transition appliquée à toutes les propriétés :

```
a {  
    transition: 1s;  
}
```

Animations



Voici deux exemples d'animation : la première utilise des pourcentages, la seconde utilise *from to*. La seconde animation est appliquée à `#blocQuiBouge` :

```
@keyframes exemple1 {
    0%   { background-color: red; }
    25%  { background-color: yellow; }
    50%  { background-color: blue; }
    100% { background-color: green; }
}

@keyframes exemple2 {
    from { left: 0px; }
    to { left: 240px; }
}

#blocQuiBouge {
    animation: exemple2 8s infinite;
}
```

@keyframes : déclare une animation (tout en lui donnant un nom).

animation : propriété générale permettant de regrouper les valeurs des sous-propriétés.

animation-name : applique une animation.

animation-duration : définit la durée de l'animation.

animation-delay : définit la durée avant le lancement de l'animation.

animation-iteration-count : définit le nombre d'itérations de l'animation (valeur *infinite* possible pour une répétition sans fin).

animation-direction : définit le sens de l'animation.

- *normal* : sens par défaut
- *reverse* : sens inverse
- *alternate* : sens alterné (changement à chaque itération)
- *alternate-reverse* : sens alterné en commençant par le sens inverse

animation-timing-function : applique une fonction sur la vitesse de l'animation.

- *ease* : commence lentement, accélère, puis décélère sur la fin
- *linear* : vitesse linéaire
- *ease-in* : commencent lentement, puis accélère
- *ease-out* : commence rapidement, puis décélère
- *ease-in-out* : semblable à *ease* mais plus modéré
- *cubic-bezier(n,n,n,n)* : pour ceux qui veulent coder leur propre courbe de vitesse

Transformations



La propriété CSS3 ***transform*** permet d'appliquer des translations, de les pivoter, de leur appliquer des homothéties, de les distordre, etc.

Quelques exemples de valeurs de la propriété ***transform*** :

- none** : pas de transformation (par défaut)
- matrix(n,n,n,n,n,n)** : Définit une matrice de transformation 2D
- matrix3d(n,n,n,n,n,n,n,n,n,n,n,n)** : Définit une matrice 4x4 de transformation 3D
- translate(x,y)** : Définit une translation 2D
- translate3d(x,y,z)** : Définit une translation 3D
- translateX(x)** : Définit une translation selon l'axe des x
- translateY(y)** : Définit une translation selon l'axe des y
- translateZ(z)** : Définit une translation selon l'axe des z
- scale(x,y)** : Définit un changement d'échelle en 2D
- scale3d(x,y,z)** : Définit un changement d'échelle en 3D
- scaleX(x)** : Définit un changement d'échelle suivant l'axe des x
- scaleY(y)** : Définit un changement d'échelle suivant l'axe des y
- scaleZ(z)** : Définit un changement d'échelle suivant l'axe des z
- rotate(angle)** : Définit une rotation 2D
- rotate3d(x,y,z,angle)** : Définit une rotation 3D
- rotateX(angle)** : Définit une rotation 3D selon l'axe des x
- rotateY(angle)** : Définit une rotation 3D selon l'axe des y
- rotateZ(angle)** : Définit une rotation 3D selon l'axe des z
- skew(x-angle,y-angle)** : Définit une rotation 2D selon les axes des x et des y
- skewX(angle)** : Définit un effet oblique 2D selon l'axe des x
- skewY(angle)** : Définit un effet oblique 2D selon l'axe des y
- perspective(n)** : Définit une vue en perspective 3D

Scroll



`scroll-behavior: smooth;`

La propriété ***scroll-behavior*** permet de configurer un défilement de la page plus doux avec la valeur ***smooth***. Les valeurs possibles sont :

- auto** : défilement par défaut
- smooth** : défilement doux

Filtres



La propriété **filter** permet d'appliquer des effets visuels (filtres) à des éléments. Les valeurs possibles sont :

none : pas de filtre (par défaut)

blur(px) : applique un flou selon une valeur donnée en pixels

brightness(%) : Ajuste la luminosité de l'image (0% : tout noir / 100% : sans modification / Au-delà de 100% : plus clair)

contrast(%) : Ajuste le contraste d'une image (0% : tout gris / 100% : sans modification / Au-delà de 100% : contraste plus élevé)

drop-shadow() : Applique une ombre

grayscale(%) : Convertit en niveau de gris (0% : sans modification / 100% : tout gris)

hue-rotate(deg) : Rotation de teinte (de 0 à 360 degrés)

invert(%) : Inverse les couleurs (0% : sans modification / 50% : tout gris / 100% : négatif)

opacity(%) : Modifie l'opacité

saturate() : Sature les couleurs (0 : sans modification)

sepia(%) : Conversion sépia (0% : sans modification)

url() : charge un filtre au format XML.

Plusieurs filtres peuvent être combinés de cette façon :

filter: blur(4px) contrast(50%);

Un filtre peut être appliqué au *background* sans impacter les contenus :

backdrop-filter: blur(10px) ;

Valeurs universelles



Presque toutes les propriétés CSS supportent ces valeurs :

initial : Applique la valeur par défaut à une propriété

inherit : Demande à hériter du parent la valeur de cette même propriété

La propriété **all** permet d'appliquer une valeur universelle à toutes les propriétés d'un élément :

all: initial;

2.7 Les propriétés de positionnement

Choisir le bon positionnement

Positionner les éléments HTML via le CSS n'est pas une mince affaire vu le nombre de possibilités. Pour tenter d'y voir plus clair, voici une petite synthèse des cas les plus fréquents et de leur solution :

- Un élément « **fixe** » est un élément qui ne défile pas avec le reste de la page : on dit qu'il est insensible au *scroll*. Il laisse les autres éléments défiler pardessous ou pardessus lui. On l'obtient avec : ***position:fixed*** et on le place en définissant ses écarts par rapport à la fenêtre : ***top, bottom, left, right***. 
- Un élément « **absolu** » recouvre les autres éléments de la page tout en défilant avec eux. On l'obtient avec : ***position:absolute*** et comme pour le précédent, on le place en définissant ses écarts par rapport à la fenêtre... ou par rapport à son premier ancêtre positionné (fixe, absolu ou relatif). Il est donc courant d'appliquer ***position:relative*** au parent d'un élément absolu pour qu'il lui serve de référence.
 - Un élément « **relatif** » peut également utiliser les propriétés ***top, bottom, left et right*** mais cette fois pour définir les écarts de l'élément par rapport à sa position d'origine. Sa position d'origine reste dans le flux, mais pas sa position finale qui peut donc recouvrir d'autres éléments...
 - Les problèmes de profondeur éventuels (par exemple un élément passant sous un autre au lieu de pardessus) se corrigent avec ***z-index*** : l'élément ayant la plus grande valeur passe pardessus les autres.
- Pour placer des blocs côte à côte, on opte soit pour un ***flex-box*** en appliquant ***display:flex*** à leur parent, soit pour des blocs-lignes avec ***display:inline-block*** ou pour des flottants avec ***float:left*** sur les éléments eux-mêmes. La solution ***flex*** comporte plus d'avantages mais n'est pas toujours nécessaire.
 - Concernant les blocs flottants (*float*), le problème de « dépassement des flottants » se corrige en plaçant un élément frère adjacent en ***clear:both***.
- Pour réserver un emplacement latéral à gauche ou à droite et laisser le reste du site se disposer autour de cet emplacement, on opte pour ***float:left*** ou ***float:right***. 
- Pour disposer des éléments selon un repère à 2 dimensions, on opte pour ***display:grid***. Les éléments conservent ainsi leurs alignements selon des lignes et des colonnes. 

Pour tous les autres éléments, et le plus souvent possible, ne codez pas de position (ce qui revient à ***position:static***) et jouez simplement sur les marges (***padding*** et ***margin***) pour ajuster leur emplacement. Même conseil pour les dimensions : codez-les le moins souvent possible (surtout la hauteur), ainsi les blocs seront libres de s'adapter.

Display : modifier le rendu CSS des éléments

Alors que la structure d'un élément HTML est non modifiable, son rendu CSS, lui, peut être modifié par la propriété ***display***. Chaque élément se caractérise donc par :

- Une appartenance à un **groupe HTML** (en-ligne ou bloc) dont dépendent les imbrications autorisées (un élément de type en-ligne ne peut pas contenir d'élément de type bloc) ;
- Un rendu CSS qui définit l'apparence globale de l'élément, son comportement, et qui est modifiable via la propriété CSS ***display***.

Cette propriété ***display*** supporte les valeurs :

- ***block*** : applique un rendu CSS de type bloc
- ***inline*** : applique un rendu CSS de type en-ligne
- ***inline-block*** : demande à un élément d'opter pour un rendu CSS bloc-en-ligne (une combinaison permettant aux éléments de s'afficher sur une même ligne tout en étant dimensionnables).
- ***none*** : demande à un élément de ne pas s'afficher (légèrement différent de ***visibility:hidden*** qui cache un élément tout en préservant son emplacement).
- ***flex***
- ***grid***

En conclusion, employez toujours les éléments selon leur fonction et non selon leur aspect (puisque cet aspect est modifiable via CSS) : un paragraphe est défini par la balise `<p>`, un titre par la balise `<h1>` ou `<h2>...`, etc. Notez que les éléments `<div>` et `` sont neutres et servent de « bouche-trou » lorsqu'aucun autre élément n'est approprié.

Avant de voir en détails ces positionnements, nous allons revenir sur la notion de **flux**.

Le flux

Le **flux** est une notion décrivant le comportement par défaut des éléments HTML. Ceux-ci, en l'absence de positionnement CSS, se placent selon l'ordre dans lequel ils sont codés, de gauche à droite, puis de haut en bas.

De plus, dans le flux, les éléments ne se chevauchent pas : chaque élément est doté d'une **consistance**, c'est-à-dire une capacité à repousser les autres éléments.

Le meilleur conseil que l'on puisse donner à un développeur débutant est de laisser un maximum d'éléments dans le flux et de n'utiliser les positions qu'en cas de nécessité.

Le positionnement normal (*position:static*)

Le positionnement normal ne nécessite pas de déclaration CSS puisqu'il s'agit du positionnement par défaut de tous les éléments HTML. Le positionnement normal respecte suiv le flux.

Au besoin, on peut tout de même appliquer ce positionnement avec la déclaration ***position:static***. Il est important de retenir qu'un élément en position statique, n'est pas affecté par les propriétés *top*, *bottom*, *left* et *right* que nous verrons par la suite.

Du rendu CSS (*display: block* et *display: inline*) d'un élément dépendent ses spécificités d'affichage :

- **Les éléments de rendu blocs se placent les uns en dessous des autres.** Par défaut, un bloc occupe automatiquement toute la largeur de son conteneur. Sa hauteur s'adapte à la hauteur de son contenu.
- **Les éléments de rendu en-lignes se placent les uns à côté des autres.** La largeur et la hauteur d'un élément en-ligne s'adaptent à son contenu.

Exemple d'éléments blocs :

`<p>Paragraphe 1</p> <p>Paragraphe 2</p>`

Ces deux paragraphes vont s'afficher l'un en dessous de l'autre, car l'élément *p* est de rendu "bloc".

Par défaut, la plupart des éléments blocs **possèdent un rendu CSS qui s'exprime par des marges internes et externes non nulles**. C'est le cas de *p*, *fieldset*, *ul*, *ol*, *h1*, *h2*, etc. dont il faudra modifier les propriétés CSS *padding* et *margin* si l'on souhaite en personnaliser les marges.

Exemple d'éléments lignes :

` Hello world `

Ce texte va s'afficher sur une seule ligne car les éléments qui le composent sont des éléments de rendu « en-ligne ». Par défaut, il n'est pas prévu que ces éléments soient positionnés sur la page (position, marges), ou dimensionnés (hauteur, largeur, profondeur).

Exemple d'éléments lignes remplacés :

``

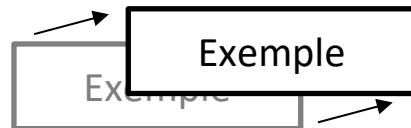
Les éléments remplacés sont des éléments dont le contenu n'est pas impacté par les styles CSS. Parmi ces éléments, nous retrouvons : *img*, *video*, *iframe*, *canvas*, *audio* et *object*. Ces éléments se comportent comme les éléments lignes, à la différence que les éléments remplacés sont positionnables et dimensionnables.

La position relative (*position:relative*)

Le positionnement relatif permet de décaler l'affichage d'un élément par rapport à son emplacement d'origine (dans le flux). Les autres contenus ne sont affectés que par l'emplacement d'origine, pas par l'emplacement final, ce qui peut entraîner des chevauchements.

Exemple d'un élément en-ligne en position relative :

```
#exemple {
    position: relative;
    bottom: 10px;
    left: 30px;
}
```



Les flottants

Les blocs flottants sont **retirés du flux normal**, et placés soit tout à droite (*float: right*), soit tout à gauche (*float: left*) de leur conteneur. Le reste du contenu s'écoule le long de ce bloc flottant, dans l'espace laissé libre.

Exemple HTML d'un bloc en position flottante :

```
<div id="conteneur">
    <div id="flottant">
        
    </div>
    La femelle du lion est la lionne...
</div>
```

Le CSS :

```
#conteneur {
    ...
}
#flottant {
    float:left;
    ...
}
```

The diagram shows a dashed-line box representing the container. Inside, there is an image of a lioness with its mouth open, and some descriptive text. Arrows point from the CSS code to the corresponding parts of the container and the floating element.

La femelle du lion est la lionne. Elle est chargée de la chasse pendant que le lion mâle surveille son territoire contre les intrusions et les menaces. Il n'existe actuellement à l'état sauvage plus que 16 500 à 30 000 spécimens dans la savane africaine.

Dans le cas de plusieurs blocs flottants à la suite, le navigateur les affiche sur une même ligne en appliquant des retours à la ligne lorsque l'espace est insuffisant.

La déclaration *float:none* permet d'annuler le positionnement flottant.

La non-consistance des flottants

Petite difficulté, les blocs flottants n'ont pas de consistance : ils n'entrent pas en compte dans le calcul de la hauteur de leur bloc conteneur.

Remarquez dans cet exemple, l'élément flottant qui semble sortir de son conteneur :

```
<div id="conteneur">
    <div id="flottant">
        Alii nullo quaerente vultus severitate adsimulata patrimonia
        sua in inmensum extollunt, cultorum ut puta feracium
        multiplicantes annuos fructus.
    </div>
</div>
```

Le code CSS :

```
#conteneur {
    ...
}
#flottant {
    float:left;
    ...
}
```

La solution est apportée par une déclaration ***clear:both*** sur un *<hr>* caché :

```
<div id="conteneur">
    <div id="flottant">
        Alii nullo quaerente vultus severitate adsimulata patrimonia
        sua in inmensum extollunt, cultorum ut puta feracium
        multiplicantes annuos fructus.
    </div>
    <hr id="correction">
</div>

#correction {
    clear:both;
    visibility:hidden;
}
```

Conclusion : La propriété ***clear*** peut prendre les valeurs *left*, *right*, *both* ou *none*. Elle se combine parfaitement avec le positionnement flottant pour corriger le problème de non-consistance des flottants. Cette propriété s'applique à n'importe quel élément, par exemple à une barre de séparation horizontale *<hr>* rendue invisible grâce à la propriété *visibility:hidden* ou *width:0*.

La position absolue (*position:absolute*)

Le positionnement absolu convient particulièrement pour les éléments ne devant pas interférer avec les autres éléments de la page (pas de collision souhaitée), comme par exemple les éléments recouvrant d'autres éléments.

Le positionnement absolu retire l'élément concerné du flux et le place selon les éloignements définis : ***top***, ***bottom***, ***left*** et ***right***. L'emplacement final ne dépend donc pas des autres éléments de la page mais uniquement des éloignements par rapport à son origine.

Les propriétés d'éloignement peuvent également prendre des valeurs négatives pour obtenir un élément dépassant de son origine.

L'origine d'un élément en position absolue est le premier ancêtre dont la position n'est pas *static*. Si l'élément en position absolue ne possède pas de tel ancêtre, il a comme origine le document (la fenêtre du navigateur).

Concernant les **dimensions**, les blocs absous se comportent comme des éléments *inline-block* : ils sont dimensionnables et ont par défaut des dimensions adaptées à leur contenu.

Dans le cas où deux propriétés d'éloignement opposées sont précisées, le bloc absolu s'étire. Par exemple, un bloc absolu avec *top:0 ; bottom:0* ; a une hauteur étirée depuis le haut jusqu'au bas de son bloc d'origine.

Exemple HTML d'un bloc en position absolue :

```
<div id="conteneur">
    <div id="absolu">
        
    </div>
    La femelle du lion est la lionne...
</div>
```

Le code CSS :

```
#conteneur {
    position: relative;
}
#absolu {
    position: absolute;
    top: 60px;
    left: 80px;
    ...
}
```

The diagram shows a dashed rectangular container labeled 'conteneur'. Inside, there is text 'La femelle du lion est la lionne...' and an image of a lion's head with its mouth open. An arrow points from the CSS rule '#conteneur { position: relative;' to the top edge of the container. Another arrow points from the CSS rule '#absolu { position: absolute; top: 60px; left: 80px; ... }' to the image. A callout box with a dashed border contains the text: 'La femelle du lion est la lionne. Elle est chargée de la chasse pendant que le lion mâle surveille son territoire contre les autres lions et menaces. Il n'existe actuellement plus que 16 500 lions dans l'Afrique. À 30 000 spécimens, la population mondiale de lionne est en Afrique.' Below the callout is a small image of a lion's head.

Dans cet exemple, seuls les éloignements *top* et *left* sont précisés. Le bloc absolu est donc placé pardessus le reste de la page, avec 60px d'écart entre le haut de l'élément et le haut de son conteneur, 80px entre la gauche de l'élément et la gauche de son bloc d'origine. Dans ce cas, le bloc absolu fait la taille de son contenu. Cela ne serait pas le cas si des éloignements opposés avaient été définis (*left* et *right* par exemple).

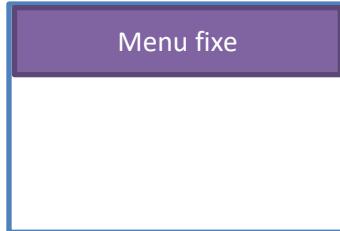
La position fixe (*position:fixed*)

Le positionnement fixe est semblable au positionnement absolu si ce n'est que **les blocs fixes restent immobiles lors du défilement de la page et ont toujours pour origine le document.**

Tout comme le positionnement absolu, le positionnement fixe est enlevé du flux et placé sur la page selon les écarts (*top*, *right*, *bottom* et *left*) définis. Ce comportement peut provoquer des chevauchements.

Exemple de code CSS :

```
.menu {  
    position:fixed;  
    left:0;  
    right:0;  
    top:0;  
}
```



Dans cet exemple, 3 écarts sont définis afin d'étendre l'élément sur toute la largeur de la page (*left:0; right:0;*) et de coller l'élément en haut de la page (*top:0;*). L'écart *bottom* n'étant pas défini, l'élément adaptera sa hauteur à son contenu.

La position *sticky* (*position:sticky*)

La position *sticky* est une curiosité permettant à un bloc de rester dans le flux (position normale) mais de se fixer lorsque l'action de *scroll* le rend hors de portée (ou plus exactement lorsque le *scroll* rend hors de portée l'emplacement qu'il aurait si les éloignements étaient appliqués à sa position d'origine).

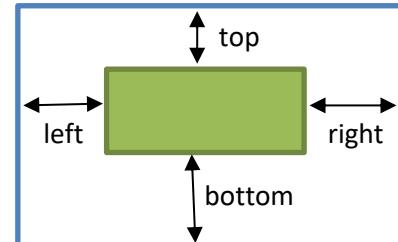
Cette position permet entre autres de coder une navigation précédée d'un header et de faire en sorte que le header disparaisse au *scroll* mais pas la navigation qui devient fixe.

Les écarts : *top*, *bottom*, *left* et *right*

Tout élément dont la position est autre que ***static*** peut se doter des propriétés *top*, *bottom*, *left* et *right* : les écarts. Retenez que sur un élément en position ***static*** (valeur par défaut), les écarts n'ont aucun effet.

Selon la position de l'élément, l'écart se réfère à différents repères :

- Dans le cas d'un élément en position ***relative***, l'écart se réfère à la position d'origine de l'élément.
- Dans le cas d'un élément en position ***fixed*** ou ***sticky***, l'écart se réfère à la fenêtre du navigateur.
- Dans le cas d'un élément en position ***absolute***, l'écart se réfère au premier ancêtre en position non ***static***, ou en l'absence d'un tel ancêtre, à la fenêtre du navigateur.



Combo !

La véritable utilité de la **position relative** est donc de contenir des blocs en **position absolue**, leur servant ainsi d'origine. Les blocs absous peuvent alors bénéficier d'une origine dépendant du flux, une accroche au contenu de la page.

Effectivement, il n'est pas toujours possible de placer un bloc absolu en se référant au document (à la fenêtre). Imaginons un site proposant une série de photos laissant apparaître des informations pardessus l'image (position absolue donc). Comme la position de chaque photo dépend des autres photos et de leur taille, difficile de connaître l'éloignement exact par rapport à la fenêtre. Dans ce cas, nous utiliserons le combo relatif + absolu.

L'ordre de superposition des éléments : *z-index*

En cas de recouvrement non désiré d'un élément par un autre, comme par exemple deux éléments en position absolues au même endroit, il est possible de gérer l'ordre dans lequel ils apparaissent pour faire en sorte que le premier recouvre le deuxième (ou l'inverse).

Il suffit pour cela d'utiliser la propriété ***z-index*** et de lui donner une valeur numérique. La plus grande valeur sera « audessus » de la plus petite. Les éléments non positionnés ont un *z-index* de 0. Une valeur négative place donc l'élément derrière les éléments non positionnés.

Exemple :

```
#cadre1 { position:absolute; z-index:4; }
#cadre2 { position:absolute; z-index:2; }
```

FlexBox

FlexBox est une solution CSS permettant de disposer des séries de boîtes (*flex-items*) dans un conteneur (*flex-container*). Les boîtes se disposent alors horizontalement ou verticalement, en se répartissant l'espace disponible avec fluidité et en passant à la ligne si l'espace est insuffisant.

Flexbox est souvent la meilleure solution pour disposer des articles dans une page, des vignettes dans une galerie, ou des liens dans une navigation.

Les FlexBox, appelées aussi **boîtes flexibles**, apportent essentiellement 4 avantages :

1. La **distribution** : les éléments sont présentés horizontalement ou verticalement (et passent de l'un à l'autre si la définition d'écran l'exige), avec passage à la ligne autorisé ou non
2. L'**alignement** : nombreuses possibilités d'alignement selon l'axe de distribution ou l'axe perpendiculaire.
3. La **réorganisation** : l'ordre des éléments est personnalisable, indépendamment du flux
4. La **fluidité** : gestion des espaces disponibles

display:flex

FlexBox se fonde sur une architecture simple :

- Un **flex-container** désigné par la déclaration CSS : **display : flex ;** (ou **inline-flex ;**)
- Des **flex-item** qui sont automatiquement les enfants du **flex-container**, sans déclaration nécessaire

Un élément *flex-item* n'est ni un bloc ni une ligne. D'ailleurs les valeurs de **display** autres que **none**, et certaines propriétés de positionnement comme **float** sont sans effet sur lui.

Pour en savoir plus sur la compatibilité de FlexBox, n'hésitez pas à consulter CanIUse.com.

flex-direction

L'axe de distribution des éléments **flex-items** se définit par la propriété **flex-direction** appliquée au **flex-container**.

Les valeurs possibles de **flex-direction** sont :

- **row** : distribution horizontale (valeur par défaut)
- **row-reverse** : distribution horizontale en sens inverse du flux
- **column** : distribution verticale
- **column-reverse** : distribution verticale en sens inverse du flux

flex-wrap

La propriété **flex-wrap** définit si les flex-items sont autorisés à passer à la ligne ou pas lorsqu'ils manquent de place. Ce passage à la ligne devient un passage à la colonne si l'axe de distribution est vertical.

Les valeurs possibles de **flex-wrap** sont :

- **nowrap** : les éléments ne passent pas à la ligne (valeur par défaut)
- **wrap** : les éléments passent à la ligne
- **wrap-reverse** : les éléments passent à la ligne dans le sens inverse

flex-flow

La propriété **flex-flow** permet de grouper les 2 propriétés précédentes en une seule.

Exemple :

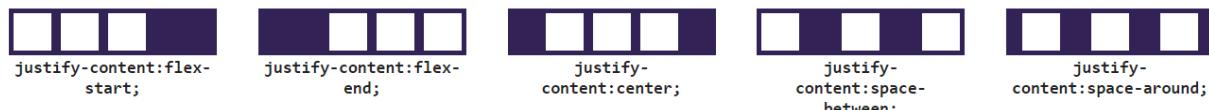
flex-flow: row wrap ;

justify-content : alignement selon l'axe principal

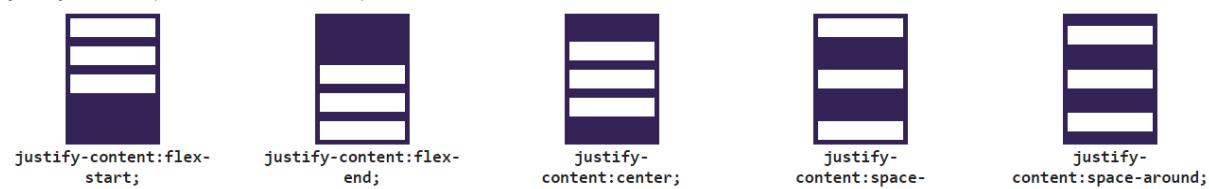
Toujours appliquée au **flex-container**, la propriété **justify-content** peut prendre les valeurs :

- **flex-start** : *flex-items* au début du sens de lecture (valeur par défaut)
- **flex-end** : *flex-items* à la fin
- **center** : *flex-items* centrés
- **space-between** : *flex-items* justifiés avec espaces entre
- **space-around** : *flex-items* justifiés avec espaces entre, et demi espaces au début et à la fin
- **space-evenly** : *flex-items* justifiés avec espaces entre, et espaces au début et à la fin

justify-content (flex-direction:row;)



justify-content (flex-direction:column;)



align-items : alignement selon l'axe secondaire

Suivant l'axe secondaire (vertical si l'axe principal est horizontal, et inversément), les alignements sont définis par la propriété **align-items**, dont les valeurs possibles sont :

- **stretch** : étirés dans l'espace disponible (valeur par défaut)
- **flex-start** : au début
- **flex-end** : à la fin
- **center** : au centre
- **baseline** : généralement identique à *flex-start*, sauf si la hauteur des *flex-items* varie

align-items (flex-direction:row;)



align-self : alignement particulier

La propriété **align-self**, permet de distinguer l'alignement d'un *flex-item* de ses frères. Les valeurs de cette propriété sont identiques à celles de **align-items** mais cette propriété s'applique cette fois au(x) **flex-items** concernés.

margin

Appliquée à des **flex-items**, la propriété **margin** permet de centrer verticalement et horizontalement avec un simple **margin:auto** ; ce qui n'est pas possible dans d'autres situations.

Il est également possible d'aligner un *flex-item* dans le bas de son *flex-container* avec **margin-top:auto** ;

order

Un autre avantage des FlexBox est de pouvoir réordonner les éléments grâce à la propriété **order**. Cette propriété a un fonctionnement semblable à celui de **z-index** : des valeurs numériques sont attribuées aux éléments et ils sont triés selon ces valeurs en ordre croissant.

flex

La propriété **flex** s'applique aux **flex-items** et comprend 3 sous-propriétés : **flex-grow**, **flex-shrink** et **flex-basis**, et dont les fonctionnalités sont :

- **flex-grow** : capacité d'un élément à s'étirer dans l'espace restant (par défaut : 0)
- **flex-shrink** : capacité d'un élément à se contracter si nécessaire (par défaut : 1)
- **flex-basis** : taille initiale de l'élément avant que l'espace restant ne soit distribué (par défaut : auto)

Par défaut, les **flex-items** n'occupent que la taille minimale de leur contenu, mais on peut rendre cette taille flexible, c'est-à-dire lui permettre de s'étendre afin d'occuper l'espace libre de son **flex-container**. Ceci se fait avec la déclaration **flex:1;** (ou **flex-grow : 1 ;**) appliquée aux **flex-items**.

Il est possible de donner des valeurs différentes aux **flex-items** pour, par exemple, qu'un élément prenne 3 fois plus de place qu'un autre : **flex:3;**

flex-grow (ou flex tout court)



Les flottants et les FlexBox

Les éléments flottants ont un comportement un peu différent lorsqu'ils côtoient des FlexBox :

- Les flottants ne débordent pas des **flex-container** ou des **flex-items**
- Les **flex-container** et les **flex-items** ne coulent pas autour des flottants (affichage de type article de journal...)
- Les marges ne fusionnent pas

Exercice

Proposez un extrait de code CSS pour obtenir le visuel souhaité



(Éléments sont étirés pour occuper toute la largeur)



(Éléments sont étirés pour occuper toute la largeur)



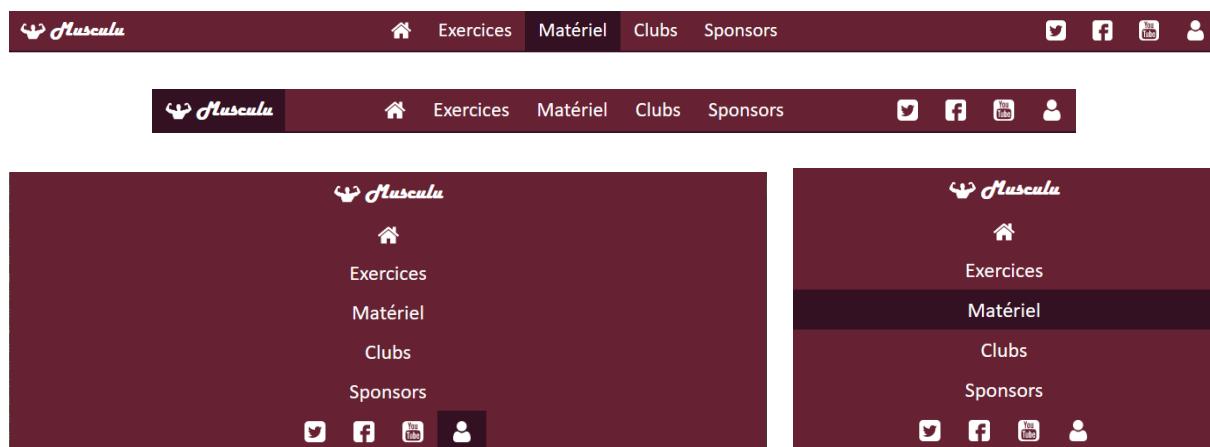
Exercice : Navigation en Flexbox

Flexbox est un positionnement CSS permettant de disposer facilement une série de blocs dans un conteneur. Le conteneur est alors appelé *flex-container* et ses enfants les *flex-items*.

Les *flex-container* peuvent s'imbriquer. C'est le cas dans cet exercice : la barre de navigation comporte 3 zones ; chaque zone comporte 1 ou plusieurs liens ; certains liens sont des icônes, d'autres du texte. À la page suivante, le code HTML de cet exercice vous est généreusement proposé. Dans ce code, 3 éléments sont en « *display:flex* » : le <header> et les 2 <nav>.

Dans cet exercice, on joue beaucoup à redimensionner la fenêtre pour observer l'élasticité des flexbox et pour provoquer le point de rupture : sous 900px, la disposition change. Pour cela, à la fin de votre CSS, utilisez l'instruction suivante et codez-y les quelques règles CSS à contredire :

```
@media screen and (max-width:900px) {  
    ...  
}
```



- Le texte est en « Calibri » et le titre « Musculu » utilise la police « Harlow Solid Italic ». Tous les textes sont de taille 1.5rem.
- Les icônes sont téléchargeables sur l'eCampus (Moodle).
- Les images (icônes) seules dans un bloc provoquent généralement un espace indésirable, qu'il est possible d'annuler avec « *display:block* ; ».
- Le logo est une image qui côtoie un texte, on peut donc ajuster son alignement vertical avec « *vertical-align* ».
- Chaque lien dispose d'un padding de « 0.5rem 1rem ».
- Les liens changent de couleur au survol, au focus et au clic. Vous pouvez voir dans les illustrations l'étendue des zones cliquables.
- Concernant les *flexbox*, nous n'utilisons dans cet exercice que les propriétés *display*, *flex-flow*, *justify-content* et *align-items*.

Voici le code HTML de l'exercice :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Flexbox exercice</title>
<style>

</style>
</head>
<body>
<header>
<a>
    <h1>
         Musculu
    </h1>
</a>
<nav id="navtop">
    <a href="#"></a>
    <a href="#">Exercices</a>
    <a href="#">Matériel</a>
    <a href="#">Clubs</a>
    <a href="#">Sponsors</a>
</nav>
<nav id="navtopright">
    <a href="#"></a>
    <a href="#"></a>
    <a href="#"></a>
    <a href="#"></a>
</nav>
</header>
</body>
</html>
```

Grid Layout

Nouveauté CSS3 aussi attendue que FlexBox, Grid Layout offre des possibilités de positionnement d'éléments en assurant un alignement à 2 dimensions (lignes et colonnes). Ce positionnement n'est pas sans nous rappeler les bons vieux <table> HTML, mais cette fois en assurant un comportement « Responsive » et d'autres avantages.

Comme pour le *flexbox*, n'importe quel élément HTML peut devenir un conteneur *grid* (une grille) avec : *display:grid* ; Les enfants de cet élément deviennent alors des cellules de la grille.

Valeurs de la propriété ***display*** :

- ***display:grid*** : pour une grille de type bloc
- ***display:inline-grid*** : pour une grille de type ligne
- ***display:subgrid*** : pour un enfant de la grille étant lui-même une grille

Autres propriétés appliquées aux grilles :

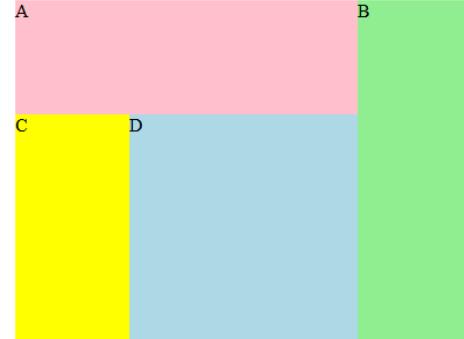
- ***grid-template-columns*** : définit le nombre de colonnes et leur largeur
- ***grid-template-rows*** : définit le nombre de lignes et leur hauteur
- ***grid-template-areas*** : permet d'associer des noms aux emplacements de cellules de la grille
- ***grid-auto-flow*** : contrôle le flux des éléments en placement automatique : ***column*** ou ***row*** (valeur par défaut distribuant les éléments de gauche à droite)
- ***gap*** : définit l'épaisseur des gouttières
- ***row-gap*** : définit l'épaisseur des gouttières horizontales (entre lignes)
- ***column-gap*** : définit l'épaisseur des gouttières verticales (entre colonnes)
- ***justify-content*** : alignement horizontal des cellules
- ***justify-items*** : alignement horizontal des éléments au sein des cellules
- ***align-content*** : alignement vertical des cellules
- ***align-items*** : alignement vertical des éléments au sein des cellules

Propriétés appliquées aux enfants des grilles (***grid-items***) :

- ***grid-row*** : indique le numéro de ligne de l'emplacement
- ***grid-row-start*** : indique le numéro de ligne du début d'emplacement
- ***grid-row-end*** : indique le numéro de ligne de fin d'emplacement
- ***grid-column*** : indique le numéro de colonne de l'emplacement
- ***grid-column -start*** : indique le numéro de colonne du début d'emplacement
- ***grid-column -end*** : indique le numéro de colonne de fin d'emplacement
- ***grid-area*** : indique le nom de l'emplacement ou 4 valeurs (par exemple 1 / 2 / 3 / 4) pour 1 : *grid-row-start* ; 2 : *grid-column-start* ; 3 : *grid-row-end* ; 4 : *grid-column-end*
- ***justify-self*** : alignement horizontal de l'élément au sein de sa cellule
- ***align-self*** : alignement vertical de l'élément au sein de sa cellule
- ***order*** : offre la possibilité de modifier l'ordre d'affichage des éléments en placement automatique

Exemple 1 :

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            main {
                display: grid;
                grid-template-columns: 100px 200px 100px;
                grid-template-rows: 100px 200px;
            }
            #A { grid-row:1; grid-column:1 / span 2;
                  background-color:pink; }
            #B { grid-row:1 / span 2; grid-column:3;
                  background-color:lightgreen; }
            #C { grid-row:2; grid-column:1;
                  background-color:yellow; }
            #D { grid-row:2; grid-column:2;
                  background-color:lightblue; }
        </style>
    </head>
    <body>
        <main>
            <div id="A">A</div>
            <div id="B">B</div>
            <div id="C">C</div>
            <div id="D">D</div>
        </main>
    </body>
</html>
```



Dans cet exemple, `<main>` devient une grille et donc ses enfants des cellules. Les cellules sont placées dans la grille en stipulant leur numéro de ligne et de colonne avec **`grid-column`** et **`grid-row`**.

La découpe de la grille est définie avec **`grid-template-columns`** et **`grid-template-rows`**. En l'absence de dimensions imposées avec *ces propriétés*, le navigateur dimensionne à sa convenance selon les contenus des cellules.

Une unité très pratique dans le cas des grilles est l'unité « **`fr`** » désignant l'espace libre (*free*) et permettant de le répartir de manière pondérée. Par exemple `grid-template-columns: 200px 1fr 2fr;` provoquera 3 colonnes respectivement de 200px, 1/3 de l'espace restant, 2/3 de l'espace restant. De plus l'unité « **`fr`** » a l'avantage de prendre en compte la largeur des gouttières, là où travailler avec des pourcentages nous amènerait des calculs complexes.

Les valeurs **`min-content`** et **`max-content`** se rapportent à la largeur (ou hauteur) de l'élément le plus petit (min) ou de l'élément le plus grand (max).

Notez également cette écriture permettant une répartition répétitive :

`grid-template-columns: repeat(3, 1fr);`

Enfin, nous pouvons automatiser un nombre de colonnes variable et leur demander une largeur entre un min et un max :

`grid-template-columns: repeat(auto-fit, minmax(120px,1fr));`

La grille est donc découpée en cellules. Les enfants de la grille, appelés aussi « grid-items » se placent dans ces cellules, et par défaut s'étirent dans ces cellules. Mais il est possible aussi que ces *grid-items* soient plus petits que les cellules et dans ce cas, il est intéressant de choisir leur alignement.

Comme pour le *flex*, on travaille avec un axe principal et un axe secondaire. Il est possible de définir l'alignement principal de tous les *grid-items* avec ***justify-items*** ou leur alignement secondaire avec ***align-items***. Mais on peut aussi appliquer à un *grid-item* particulier les propriétés ***justify-self*** ou ***align-self***. Les valeurs possibles pour ces 4 propriétés sont :

- ***start*** : aligne au début de la cellule
- ***end*** : aligne à la fin de la cellule
- ***center*** : centre l'élément
- ***stretch*** : étire l'élément pour occuper toute la cellule

Exemple 2 :

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            main {
                display: grid;
                grid-template-areas: "haut haut"
                                     "gauche droite"
                                     "bas bas";
            }
            #A {
                grid-area:haut; background-color:pink;
            }
            #B {
                grid-area:gauche; background-color:lightgreen;
            }
            #C {
                grid-area:droite; background-color:yellow;
            }
            #D {
                grid-area:bas; background-color:lightblue;
            }
        </style>
    </head>
    <body>
        <main>
            <div id="A">A</div> A
            <div id="B">B</div> B
            <div id="C">C</div> C
            <div id="D">D</div> D
        </main>
    </body>
</html>
```

Dans cet exemple, une propriété ***grid-template-area*** permet de décrire l'emplacement des cellules. On travaille ici avec des chaînes de caractères (au choix) qui sont ensuite appliquées aux cellules avec ***grid-area***. Remarquez que l'on peut obtenir des « fusions » de cellules en répétant la chaîne.

Exercice : *Fandom* en Grid Layout

Choisissez rapidement un film, autre que celui de l'exemple. Récoltez 6 acteurs (nom, rôle et photo), une image horizontale, un synopsis d'environ 2 paragraphes et quelques détails techniques (réalisateur, producteur, scénariste, sortie du film, langue originale, durée, etc.)

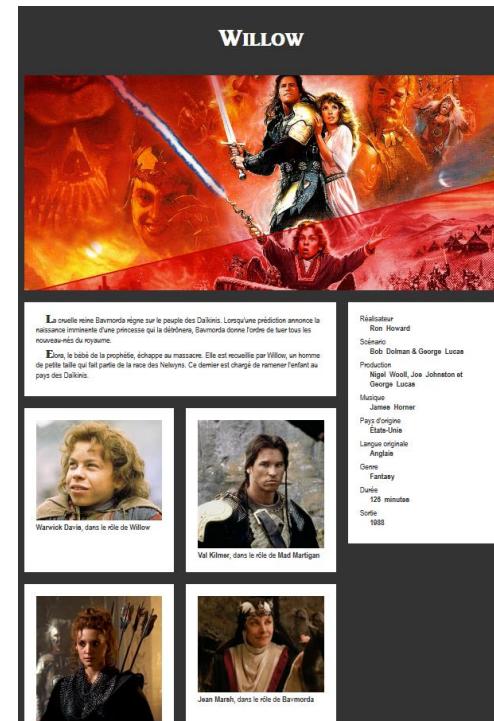
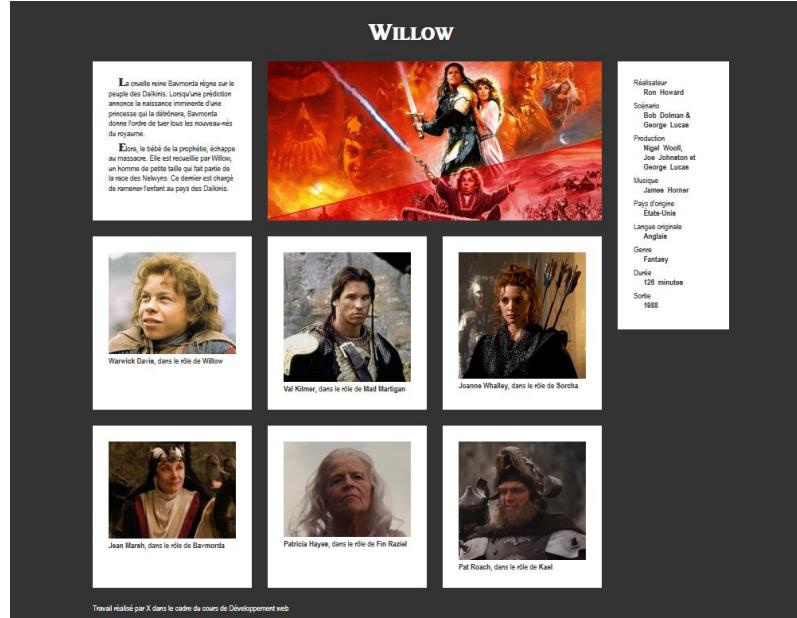
Cet exercice porte sur l'utilisation de Grid Layout afin de découper une page web en grille à 2 dimensions. De plus, 3 points de rupture (*media-queries*) sont définis afin de s'adapter à différentes résolutions d'écran : 500px, 900px et 1300px. N'oubliez pas de coder une méta-balise *viewport*.

La grille est un élément `<main>` centré horizontalement et dont la largeur maximum (*max-width*) est de 1600px. Cette grille comporte 11 cellules : 8 avec un fond blanc, 1 avec une image en background et 2 avec fond transparent (le titre et le pied de page).

La cellule de titre est `<h1>`, la cellule de pied de page est `<footer>`, la cellule de détails est `<aside>`, les cellules d'acteurs sont des `<figure>` et les autres des `<div>`.

Concernant la cellule avec l'**image du film**, utilisez un *background-image* en CSS ainsi qu'une hauteur minimale parce que cette cellule est dépourvue de contenu et risque d'avoir une hauteur de 0. Concernant les **détails** du film, utilisez une liste `<dl>`. Concernant les **photos des acteurs**, choisissez des images plus ou moins carrées d'au moins 300px de large et appliquez-leur une largeur de 100%.

Le comportement par défaut présente 4 colonnes, des gouttières de 40px et des marges intérieures de cellules de 40px. La 4^e colonne fait 280px et les autres se répartissent équitablement la largeur de la grille... mais n'utilisez pas l'unité « % » car une autre unité est plus indiquée dans le cas de gouttières non nulles !



Sous 1300px, la grille passe en 3 colonnes de tailles égales et les gouttières et marges intérieures des cellules passent à 30px.

Sous 900px, la grille passe en 2 colonnes et les gouttières et marges intérieures des cellules passent à 20px.

Sous 500px, la grille passe en 1 colonne et les gouttières et marges intérieures des cellules passent à 10px.

La propriété *align-self* est utilisée sur la cellule de détails afin que cet élément ne soit pas étiré sur toute la hauteur de sa cellule lorsqu'il y a 3 ou 4 colonnes.

Pour le référencement, agrémentez votre code HTML de métabases *description* et *keywords*.

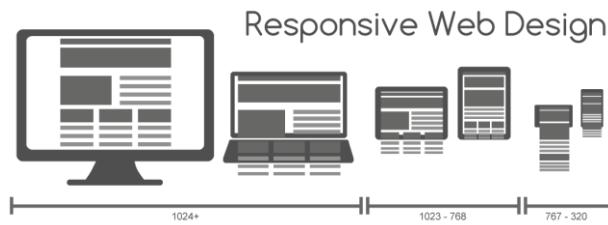
Dans les détails, utilisez l'astuce de l'espace insécable entre les noms et prénoms pour ne pas qu'un retour à la ligne ne survienne entre les 2.

Pour finir, appliquez une police au contenu de la page (par exemple Arial, Arial Narrow, Calibri, Cambria, etc.) et une autre police plus exotique pour le titre et les lettrines du synopsis. Si le titre est plus long que dans l'exemple, n'hésitez pas à travailler son *font-size* dans chaque *media-query*.



2.8 Responsive Design

L'agencement des informations dans un site, en tenant compte de la grande variété de définitions d'écran, s'est complexifié avec l'arrivée des smartphones et des tablettes. L'ancienne solution, consistant à développer un site dont le contenu se limitait à un bloc centré d'une largeur définie (par exemple 960px), ne convient plus à la situation actuelle.



Dorénavant, la conception d'un site web *responsive* nécessite d'imaginer l'agencement des informations sur des tailles d'écran allant de 320px à 1920px ou plus, ce qui nécessite le réagencement ou l'élasticité des éléments.

Mais sommes-nous obligés de développer *responsive* ? Quelles sont les autres solutions ?

Statique, fluide, adaptatif ou *responsive* ?

Au sens large du terme, un site *responsive* est donc un site qui s'adapte à toutes les définitions d'écran à partir de 320px. Mais si nous souhaitons être rigoureux, il convient de distinguer les **designs statiques, fluides, adaptatifs et responsive**.

Le design statique (ou fixe) : se réfère à des dimensions figées (souvent définies en pixels) quelle que soit la surface de l'écran. La grande majorité des sites web étaient construits sur cette base avant l'arrivée du *Responsive Web Design* dans les années 2010.

Le design adaptatif : amélioration du design statique où les unités de largeur sont fixes, mais diffèrent selon la taille de l'écran, car des *media-queries* CSS testent la largeur de l'écran.

Le design fluide (ou liquide) : site web dont toutes les largeurs sont exprimées en unités variables (% , vw, vh, etc.) afin de s'adapter à chaque taille de fenêtre, jusqu'à une certaine mesure.

Le design responsive : est une combinaison judicieuse de design fluide et adaptatif. Les conteneurs principaux s'adaptent proportionnellement à toutes les dimensions (=fluide) et certains points de rupture provoquent des changements significatifs d'agencement de contenus (=adaptatif), comme des blocs côté à côté qui passent l'un en dessous de l'autre, ou un agencement en 4 colonnes qui passe à 2 colonnes.

Les définitions d'écran

Dorénavant, en Belgique, la majeure partie des visites de sites web se font depuis un mobile. Cette tendance est encore plus élevée dans les autres pays européens et dans le monde.

Les définitions d'écrans (souvent appelées résolutions d'écran à tort) sont nombreuses et varient en largeur depuis 320px jusqu'à 2560px. Les définitions les plus courantes sont : **320x480 ; 320x534 ; 320x568 ; 360x640 ; 375x667 ; 412x732 ; 412x846 ; 414x736 ; 600x1024 ; 601x962 ; 720x1280 ; 768x1024 ; 800x1280 ; 1024x600 ; 1280x800 ; 1280x1024 ; 1366x768 ; 1440x900 ; 1536x864 ; 1600x900 ; 1920x1080**.

Le viewport

La métá-balise *viewport* qui permet aux mobiles d'appliquer la définition apparente :

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Les media-queries

Voici un exemple de *media-query* CSS qui permet de détecter les « points de rupture ». Prévoyez-en autant que souhaité. Il n'est pas question de recoder tout le CSS dans chaque *media-query* mais de ne coder que les instructions contredisant le CSS principal.

```
@media screen and (max-width: 760px) {  
    #main { width:760px ; }  
    ...  
}
```

Les solutions pratiques pour un comportement *Responsive*

Les textes et les éléments lignes d'enrichissement des textes

Les textes et leurs éléments lignes d'enrichissement comme ``, `<i>`, ``, ``, `<u>`, ``, `<s>`, `<sup>`, `<sub>`, etc. ont par défaut un comportement adaptatif puisque chaque espace est un possible retour à la ligne en cas de place insuffisante.

Malgré cela, il arrive qu'un mot récalcitrant dépasse de son conteneur. Soit parce qu'il est très long (comme une URL ou « anticonstitutionnellement »), soit parce que la police est un peu grande. La première solution est de régler la taille de la police en prévoyant un comportement adaptatif :

```
h1 { font-size:3rem; }  
@media screen and (max-width:640px) {  
    h1 { font-size:2.5rem; }  
}  
@media screen and (max-width:480px) {  
    h1 { font-size:2rem; }  
}
```

Il arrive qu'un mot très long dépasse de son conteneur. En l'absence d'espaces dans le mot, celui-ci ne sera pas coupé pour passer à la ligne. Deux solutions s'offrent à vous : cacher ce qui dépasse ou forcer la césure au sein des mots trop longs.

```
... { overflow:hidden ; }  
... { overflow-wrap:break-word ; }
```

Les éléments lignes « remplacés »

Certains éléments en-ligne sont dits « remplacés » car occupent un espace « remplacé » par un objet. Il s'agit des éléments ``, `<video>`, `<iframe>`, `<input>`, `<textarea>`, `<select>`, `<button>`, `<audio>`, `<canvas>`, `<hr>`, `<svg>` et même `
`.

Par défaut, un élément remplacé, comme une image, a une taille fixe. Cette taille, si elle dépasse 320px de large, peut poser des problèmes d'adaptabilité. La solution est d'imposer, en plus de la taille, une taille maximale en % qui ne sera jamais dépassée par l'élément :

```
img, video, iframe, input { max-width:100% ; }
```

Les background-image

Les images de fond (*background-image*) peuvent facilement adapter un comportement *responsive* avec la règle *background-size*, qui peut prendre par exemple comme valeur *cover* ou *contain* :

```
background-size:cover;
```

Cette règle va adapter les dimensions de votre image de fond à celles de son élément, mais il est aussi souvent apprécié de centrer cette image et ne pas la répéter :

```
background-position:center center;  
background-repeat:no-repeat;
```

Les éléments bloc

Lorsque vous codez le CSS de votre site, évitez de préciser des dimensions si ce n'est pas nécessaire. Laisser libres les hauteurs de vos éléments leur permettra de s'adapter si nécessaire !

Privilégiez **width:auto** ; plutôt que **width:100%** ; qui a tendance à dépasser si l'élément comporte un *padding* ou un *border*.

```
... { width:auto ; }
```

Comme pour les images, une largeur maximale de 100% évitera des dépassements inélégants :

```
... { max-width:100% ; }
```

Pour les éléments positionnés en absolu ou en fixe, n'oubliez pas que vous pouvez préciser à la fois l'écart gauche et droit de l'élément pour l'étendre sur toute la surface du site et vous assurez qu'il ne dépasse pas.

```
#popup { position:fixed; left:2vw; right:2vw; }
```

Dès éléments blocs côte à côte seront sans doute plus à leur aise les uns en dessous des autres sur de petits écrans :

```
article { display:inline-block; }  
nav { float:left; }  
@media screen and (max-width: 640px) {  
    article { display:block;}  
    nav { float:none; }  
}
```

Enfin, il existe 2 solutions pour positionner les éléments blocs tout en leur assurant un comportement *responsive* : flexbox et grid-layout.

2.9 L'accessibilité du Web

Petit historique et législation actuelle

En 1996, année de sortie du langage CSS, le *World Wide Web Consortium* (W3C) crée la **Web Accessibility Initiative (WAI)** qui regroupe des experts issus de l'industrie technologique, des organismes pour personnes handicapées, de la recherche et des gouvernements. L'objectif de la WAI est de garantir l'accès au Web par les personnes handicapées.

En 1999, la WAI présente la version 1.0 de ses recommandations pour l'accessibilité : les **Web Content Accessibility Guidelines**, ou **WCAG** (prononciation « wécague ») : une cinquantaine de critères classés en 3 niveaux nommés « A », « AA » et « AAA ». Les WCAG deviennent la norme internationale de référence pour l'accessibilité du Web.

Au début des années 2000, plusieurs pays votent des lois obligeant les services publics à appliquer les WCAG. L'effet de ces lois n'est cependant pas un succès : les critères sont méconnus des concepteurs et développeurs, et représentent une surcharge de travail considérable. Le Web accessible reste malheureusement une utopie.

En 2008, les WCAG 2.0 deviennent la nouvelle norme internationale, suivie en 2018 par les WCAG 2.1.

En 2016, la directive européenne sur l'accessibilité des sites web entre en vigueur et une version légèrement modifiée est transposée en Fédération Wallonie-Bruxelles en 2019. Cela signifie que tous les organismes publics belges (gouvernements, transports en commun, institutions, communes, CPAS, etc.) doivent dorénavant présenter des sites web, applications mobiles, contenus vidéo ou audio et documents téléchargeables, de manière conforme aux WCAG 2.1 niveau AA.

En Belgique, AnySurfer (<https://www.anysurfer.be/fr>) est la référence en termes de documentation, de certification et de formation à l'accessibilité. Les certifications sont délivrées pour une durée de vie de 2 ans.

L'accessibilité

L'un des grands défis pour l'évolution du Web est de le rendre accessible aux personnes qui ont des besoins spécifiques en termes d'interface. C'est le cas des personnes en situation de handicap : handicap visuel (daltonien, non-voyant ou malvoyant), handicap auditif, handicap moteur ou handicap mental.

Plus largement, l'accessibilité aux contenus web concerne également les handicaps techniques : le contenu doit être accessible sur tous les périphériques connectés au Web, qu'il s'agisse de technologies anciennes ou récentes.

Président du W3C, Tim Berners-Lee déclare :

« À travers tout notre travail sur les langages hypertextes, graphiques et multimédia, on retrouve notre préoccupation d'un accès pour tous à l'information indépendamment de la culture, du langage et du handicap. La WAI conçoit des protocoles et des logiciels qui rendent le Web accessible aux personnes atteintes de handicaps auditifs, physiques, cognitifs ou neurologiques. (...) L'essentiel de cet effort là n'est effectif que si les concepteurs web prennent en compte l'accessibilité dans le cadre de leur travail. »

Comment les CSS améliorent l'accessibilité ? La démarche proposée par le W3C est en fait très simple, et consiste à **séparer la structure du document de sa présentation**. Le document est structuré indépendamment du périphérique de sortie, alors que les feuilles de styles CSS peuvent changer selon le périphérique de sortie : écrans, synthétiseurs vocaux, impression, imprimantes Braille, écrans de télévision, projecteurs, etc.

Le document HTML n'utilise plus de balises pour la présentation, mais uniquement pour la sémantique.

Les handicaps concernés

Handicap moteur

Ce groupe est composé d'une grande variété de handicaps. C'est lorsque l'handicap moteur concerne les membres supérieurs que l'impact sur la navigation se fait ressentir. Dans beaucoup de cas, l'utilisation d'une souris ou d'un clavier est très difficile, voire impossible. Ces personnes utilisent souvent des navigateurs conventionnels, mais leur principal problème est la navigation dans la page, utilisant l'équivalent de la touche Tabulation pour accéder en séquence à tous les éléments cliquables.

Handicap visuel : non-voyants

Il s'agit du handicap le plus souvent pris en compte, grâce au travail remarquable d'associations comme BrailleNet. L'utilisateur aveugle peut utiliser un navigateur moderne ou texte associé à un synthétiseur vocal aussi appelés lecteurs d'écran (IBM Home Page Reader, JAWS), une plage Braille (qui restitue sous forme de matrice de picots les caractères Braille). Une des principales difficultés est la navigation, car le non-voyant navigue dans la page de façon séquentielle.

Handicap visuel : malvoyants

Pour des handicaps visuels légers, les navigateurs permettent de zoomer facilement les pages (ctrl+), de configurer la taille du texte ou de choisir la police par défaut. Si l'handicap est plus prononcé, la loupe du système d'exploitation peut prendre le relais.

Handicap visuel : daltoniens

Une catégorie à part parmi les malvoyants concerne les daltoniens, personnes ayant des difficultés à différencier certaines couleurs. Les daltoniens sont spécifiquement générés dans deux situations : lorsque la couleur d'un texte se confond avec la couleur de fond. La solution est de choisir des contrastes clair/obscur plutôt que des contrastes de teinte ; lorsqu'une partie de l'information est codée uniquement par la couleur. Un simple indice en texte permet alors de contourner le handicap.

Handicap auditif

Ce groupe est l'un des moins atteints, dans la mesure où son principal souci est de percevoir des signaux audios comme les bips de message d'erreurs ou la bande-son d'une animation multimédia. Quand le handicap auditif s'additionne au handicap visuel, la plage Braille est bien souvent l'unique solution adaptative.

Handicap cognitif ou neurologique

C'est certainement le groupe le plus difficile à contenter, avec des handicaps très légers comme la dyslexie, jusqu'à des problèmes beaucoup plus graves. Pour les personnes lisant lentement, le défilement de texte peut être trop difficile à lire. Pour les épileptiques, certains types de clignotement sont néfastes et peuvent provoquer des crises.

Résumé des critères d'accessibilité



Pour une information exhaustive, visitez la documentation officielle : <https://www.w3.org/TR/WCAG21/>

- Les images présentent des textes alternatifs, exception faite des images de décoration n'illustrant pas le sujet de la page :
- Les intitulés de bouton de formulaire décrivent leur fonction : <input value="...">
- Les champs de formulaire sont associés à des étiquettes : <label for="...">
- Les contenus audio et vidéo sont accompagnés de retranscription textuelle, d'audiodescription et de sous-titres.
- Un mécanisme est disponible pour mettre en pause, pour arrêter ou contrôler le volume du son qui démarre automatiquement sur une page pendant plus de 3 secondes.
- Les contenus sont encadrés de balises sémantiques adaptées.
- L'ordre de lecture et de navigation est logique et intuitif. Il ne change pas d'une page à l'autre.
- Les instructions ne dépendent pas d'un son.
- Les instructions ne dépendent pas de la forme, de la taille ou du positionnement. Par exemple : « cliquez le carré à droite »...
- L'orientation de la page web s'adapte au mode portrait comme au mode paysage.
- La couleur n'est pas la seule façon de distinguer un lien du texte qui l'entoure, sauf si le contraste entre les 2 est d'au moins 3:1 et qu'une autre manière de différencier le lien (soulignement par exemple) est utilisée lors du survol et du focus.
- Le texte a un contraste d'au moins 4,5:1 avec le fond. Toutefois, les textes de grande taille (24 pixels ou 18,66px + gras) peuvent présenter un contraste de 3:1.
- La page est lisible et fonctionnelle lorsque zoomée à 200%.
- S'il est possible d'obtenir la même présentation visuelle en utilisant du texte plutôt qu'une image, alors cette image doit être remplacée par un texte.
- Il n'y a pas de pertes de contenus ou de fonctionnalités et le défilement horizontal est évité lorsque le contenu est présenté sur une largeur de 320 pixels.
- Aucune perte de contenu ou de fonctionnalité ne se produit lorsque l'utilisateur adapte la hauteur et l'espacement de la ligne de texte à 1,5 fois la taille de la police, l'espacement des paragraphes à 2 fois la taille de la police, l'espacement entre les mots à 0,16 fois la taille de la police et l'espacement entre les lettres à 0,12 fois la taille de la police. D'où l'importance de ne pas figer la hauteur des conteneurs...
- Toutes les fonctionnalités d'une page sont utilisables à l'aide du clavier.
- Tout contenu en mouvement, clignotant ou défilant pendant plus de 5 secondes peut être mis en pause, arrêté ou masqué par l'utilisateur.
- Aucun contenu de la page ne flashe plus de 3 fois par seconde.
- La page web a un titre informatif et descriptif.
- L'ordre de tabulation des liens, des éléments de formulaire, etc. est logique et intuitif.
- La fonction de chaque lien peut être déterminée à partir du texte du lien et de son contexte.
- Il existe plusieurs manières de trouver d'autres pages sur le site (plusieurs navigations, plan du site, barre de recherche, etc.).
- Il est possible de distinguer visuellement sur quel élément se trouve le focus du clavier.
- La langue de la page est indiquée avec l'attribut *lang*. Si un contenu est dans une autre langue, l'attribut *lang* indique la langue de ce contenu.
- La réception du focus par un élément ne provoque par l'apparition de pop-up ou d'autres modifications perturbantes.
- Les champs de formulaire obligatoires présentant des règles (longueur maximale par exemple), présentent ces règles dans leur élément <label>.
- Les erreurs significatives de validation syntaxique sont évitées => Validator !

2.10 Les problèmes de compatibilité et leurs solutions

Il existe un décallage entre la rédaction des normes officielles HTML/CSS et l'implémentation de celles-ci dans les navigateurs. Certaines équipes de développement sont plus rapides que d'autres et toutes les équipes ne travaillent pas dans le même ordre. Si bien que la compatibilité peut varier pour chaque instruction HTML ou CSS.

Actuellement, ce sont les normes **CSS3** qui posent le plus de problèmes de compatibilité, car certaines sont très récentes et que leur support par les navigateurs évolue sans cesse. Avant d'utiliser une propriété CSS3, il convient donc de vérifier sa compatibilité.

Les préfixes CSS

À propos de la propriété ***border-radius***, voici ce que nous apprend le site www.w3schools.com :

Property					
border-radius	5.0 4.0 -webkit-	9.0	4.0 3.0 -moz-	5.0 3.1 -webkit-	10.5

On constate ici que les navigateurs Chrome, Firefox et Safari ont développé un support partiel en attendant de développer le support intégral. Par exemple, le préfixe -webkit- permet à Chrome 4.0 d'accéder au support partiel, alors que Chrome 5.0 utilisera le support complet (sans préfixe).

Le support partiel diffère d'un navigateur à l'autre et d'une propriété à l'autre. Il pourrait par exemple s'agir de bordure arrondie qui ne supporte que les arrondis circulaires et non les ellipses...

D'autres navigateurs comme Edge et Opera ne sont pas passés par un support partiel de la propriété ***border-radius***. Il ne sert donc à rien d'utiliser -ie-border-radius ou -o-border-radius : cela n'existe pas.

Dans le cas de ***border-radius***, pour obtenir un support maximal, on codera :

```
... {
    -webkit-border-radius:8px ;
    -moz-border-radius:8px ;
    border-radius:8px ;
}
```

De cette manière, les visiteurs utilisant Firefox 3.6 pourront profiter de belles bordures arrondies. Dans la pratique, on prend rarement autant de précaution pour une règle comme ***border-radius***, mais si la règle est plus importante ou plus récente, on peut se donner la peine de coder les règles préfixées.

Remarquez que l'on code toujours les règles avec préfixe avant les règles sans préfixes !

Gardez à l'esprit qu'une propriété non compatible peut tout de même être utilisée pour des actions non critiques de votre site comme des mouvements, des effets décoratifs ou des interactions, tant que l'accessibilité et l'utilisabilité de votre site n'en est pas affectée. Il n'est donc pas interdit de coder simplement :

```
... {
    border-radius:8px ;
}
```

Les commentaires conditionnels



EN HTML, il est possible de créer des commentaires particuliers qui ne seront interprétés que par Internet Explorer :

<!--[if lte IE 6]> pour IE6 et versions antérieures <![endif]-->
 <!--[if lt IE7]> pour les versions antérieures à IE7 <![endif]-->
 <!--[if !IE]> pour les navigateurs différents d'Internet Explorer <![endif]-->

Opérateur	Signification
!	« n'est pas »
lt	« inférieur à » ou « Lower Than »
lte	« inférieur ou égal à » ou « Lower Than or Equal »
gt	« supérieur à » ou « Greater Than »
gte	« supérieur ou égal à » ou « Greater Than or Equal »

Ces commentaires conditionnels peuvent par exemple être utilisés pour insérer une feuille de style complémentaire pour parer aux bugs d'IE.

Les hacks CSS



Une autre façon d'imposer des règles CSS à certains navigateurs et pas à d'autres est d'utiliser des sélecteurs CSS exploitant des particularités dans le comportement des navigateurs.

Hack CSS	Navigateurs visés
* html { }	IE6-
*html {}	IE7-
*+html {}	IE7-
html > body { }	Navigateurs récents et IE7+
html>/**/body { }	Navigateurs récents et IE8+



2.11 Quelques outils CSS

Les *reset CSS*

Le *reset CSS* est un fichier CSS (ou un ensemble de règles CSS) imposant une valeur nulle à bon nombre de propriétés CSS sur certains types d'éléments. Cette technique permet d'éviter certaines différences d'affichage entre navigateurs.

Un fichier CSS commence souvent par un *reset CSS*.

Exemple : le *reset CSS* d'Éric Meyer : <https://meyerweb.com/eric/tools/css/reset/>

Feuille de styles de base

Évolution du *reset CSS*, la feuille de style de base corrige l'affichage par défaut des éléments HTML en appliquant des valeurs non nulles aux propriétés CSS de certains types d'éléments.

Cet outil permet à la fois de gommer les différences entre navigateurs mais aussi de fournir un canevas : une base mise en page.

Exemple : KNACSS : <https://www.knacss.com/>

Sass et Less

Sass est un langage de script préprocesseur qui est compilé ou interprété en CSS. SassScript est le langage de script en lui-même. Sass se compose de deux syntaxes. La syntaxe originale, appelé "la syntaxe indentée" utilise l'indentation pour séparer les blocs de code et les sauts de ligne pour les séparer des règles.

Less est un langage dynamique de génération de CSS. Initialement inspiré par Sass, il l'influence à son tour avec l'apparition de la syntaxe « SCSS » par laquelle Sass reprend des éléments de la syntaxe CSS classique. Le principe de Less est en effet de ne pas rompre avec la syntaxe CSS : tout code CSS est aussi du code Less valide et sémantiquement équivalent.

2.12 Exemples HTML+CSS

Exemple : un contenu qui chevauche un autre



Okko : bande dessinée fantastique

Une invitation au voyage et au dépaysement, au temps du Japon féodal : Okko, le rônin sans maître, est à la tête d'un groupe de chasseurs de démons et arpente les terres de l'empire du Pajan (version médiévale-fantastique du Japon) en proie à la guerre.

Dans cet exemple, un bandeau étalé sur toute la largeur de la page est chevauché par le bloc de contenu.

Un tel chevauchement peut être obtenu avec des positions (relative, absolue, etc.), mais il existe aussi un moyen plus simple dans ce cas : appliquer une marge négative au bas du bandeau.

Un peu d'HTML :

```
<header id="bandeau"></header>
<main>
    <h1>Okko : bande dessinée fantastique</h1>
    <p>Une invitation au voyage...</p>
</main>
```

Et le CSS :

```
header#bandeau {
    background-image:url(img/okko-bandeau.jpg);
    background-position:center center;
    background-size:cover;
    height:50vh;
    margin-bottom:-10vh; ←
}

main {
    width:80vw;
    margin:0 auto;
    padding:2vw;
    background-color:#fff;
    border-radius:0.25rem;
}

main h1 {
    border-left:0.25em #f60 solid;
    margin:2em 0 1em 0;
    padding:0.5em 1em;
}
```

L'astuce de la marge négative : cela permet à l'élément suivant de remonter.

Cette astuce a tout de même ses limites : on ne peut pas choisir qui chevauche qui. Pour modifier l'ordre de superposition, il faudrait utiliser la propriété *z-index* appliquée à un élément en position non statique (par exemple *position:relative*).

Exemple : une disposition texte + image en 50/50

Okko : bande dessinée fantastique

Une invitation au voyage et au dépaysement, au temps du Japon féodal : Okko, le rônin sans maître, est à la tête d'un groupe de chasseurs de démons et arpente les terres de l'empire du Pajan (version médiévale-fantastique du Japon) en proie à la guerre.

Redoutable mercenaire, Okko est accompagné de Noburo, colosse au masque rouge, de maître Noshin, bonze fantasque, et du

jeune Tikku, pêcheur devenu membre à part entière du petit groupe à qui Noshin enseigne son art d'invoquer et de communiquer avec les forces de la nature.

La série éditée chez les éditions Delcourt, est aujourd'hui terminée et compte 10 tomes répartis en 5 cycles. Elle s'est vendue à plus de 1 300000 exemplaires pour l'ensemble des éditions francophones et a été traduite en une dizaine de langues.



Cette disposition en 50/50 présente un texte à côté d'une image. Ce type de disposition varie les plaisirs visuels, tout en évitant de grands espaces vides et des lignes de textes trop longues.

```
<section class="dispo50_50">
    <div class="left">
        <h2>Okko : bande dessinée fantastique</h2>
        <div class="columns">
            <p>Une invitation au voyage ...
            <p>Redoutable mercenaire ...
            <p>La série éditée ...
        </div>
    </div>
    <div class="right">
        
    </div>
</section>
```

```
/* Disposition 50/50 et texte en 2 colonnes */
.dispo50_50 { display:flex; justify-content:space-between; }
.dispo50_50>* { width:48%; }
.dispo50_50>.right {
    display:flex;
    justify-content:center;
    align-items:center;
}
/* Texte en 2 colonnes */
.columns { column-count:2; }
.columns p:first-child { margin-top:0; }

@media screen and (max-width:1050px) {
    .dispo50_50 { flex-direction:column; }
    .dispo50_50>* { width:100%; }
}

@media screen and (max-width:788px) {
    .columns { column-count:1; }
}
```

Les enfants du *flexbox* font 48% de large. Grâce à l'alignement en *space-between*, les 4% restant forment une goutière entre les 2.

Le *flexbox* sur l'élément *right* ne sert qu'à centrer horizontalement et verticalement l'image.

La marge top du 1^{er} paragraphe provoquait un désalignement des 2 colonnes de texte.

Sur des écrans plus étroits, on fait passer l'image sous le texte, puis passer le texte en une colonne unique.

Exemple : des grands pavés de navigation (image + texte)

Les personnages



Dans cet exemple, une série de pavés disposés horizontalement présentent chacun une image et un texte. Difficulté supplémentaire : ils sont cliquables. **Attention, dans ce cas de figure, il faut toujours s'assurer que toute la surface du pavé soit cliquable !**

Dans cet exemple, afin d'inviter au clic, le survol des pavés déclenche un effet de zoom (le pavé grossit). **Attention aux changements de dimensions lors du survol : cela ne doit pas bousculer les autres éléments de la page !**

```
.paves {
  display:flex;
  flex-flow:row wrap;
  margin:-2%;
}

.paves a {
  margin:2%;
  width:21%;
  text-decoration:none;
}

.paves a figure {
  margin:0;
  transition:400ms;
  background-color:#840;
  color:#fff;
}

.paves a:hover figure {
  transform:scale(1.2);
}

.paves a figure img {
  width:100%;
  display:block;
}

.paves a figcaption {
  padding:1rem 2rem;
}
```

Flexbox est la meilleure solution pour disposer des blocs côté à côté.

Pour un alignement parfait avec le titre, cette marge négative compense les 2% de marge des <a>,

Pour en aligner 4 : 25% moins 2 fois les marges.

Au survol du <a>, son <figure> s'étire et dépasse du <a>, mais sans bousculer les autres <a>.

Les images s'étendent sur toute la largeur de <figure>. La hauteur libre, les proportions sont conservées.

Sans le *display:block*, les images provoquent un petit espace...

```
<div class="paves">
  <a href="#">
    <figure>
      
      <figcaption>Okko</figcaption>
    </figure>
  </a>
  <a href="#">
    <figure>
      
      <figcaption>Noburo</figcaption>
    </figure>
  </a>
  <a href="#">
    <figure>
      
      <figcaption>Noshin</figcaption>
    </figure>
  </a>
  <a href="#">
    <figure>
      
      <figcaption>Tikku</figcaption>
    </figure>
  </a>
</div>
```

Exemple : la technique des *sprites*

Le terme *sprite*, dérivé du latin *spiritus*, désigne un lutin. Dans l'univers de l'infographie et des jeux vidéo, les *sprites* sont des images 2D souvent en partie transparentes. Elles peuvent représenter des personnages, des projectiles, des explosions, ou d'autres éléments d'un jeu.



Dans le cas d'une animation, le fichier image du *sprite* contient plusieurs fois le personnage avec de légères variations de position. Le passage éclair d'une position à l'autre donne une impression de mouvement.

Dans l'univers du Web, cette pratique a été reprise pour créer des images de boutons selon plusieurs états. Au survol ou au clic du bouton, le CSS se charge de décaler le *sprite*.



Pour cela, nous codons par exemple un bouton avec un simple <div> :

```
<div class="bouton">Bouton</div>
```

En CSS, nous appliquons l'image en *background* :

```
.bouton {
    font-family:Arial, sans-serif;
    background-image:url(button-sprites.png);
    background-position:0 0;
    width:131px;
    height:26px;
    padding:8px 16px;
    color:#fff;
    cursor:pointer;
}
.bouton:hover, .bouton:focus {
    background-position:0 -47px;
}
.bouton:active {
    background-position:0 -94px;
}
```

La position du *background* se décale lors du survol et du clic du bouton.

Exemple : personnaliser les boutons *radio* ou *checkbox*

Le visuel classique des boutons *radio* et *checkbox* ne plaît pas à tout le monde. Il est possible d'en modifier l'apparence tout en conservant leur fonctionnalité.



Pour cela, nous accompagnons chaque bouton d'un <label>, dont l'attribut *for* spécifie l'*id* du bouton connecté.

```
<input type="checkbox" name="options[]" class="options" id="option1">
<label for="option1">Option 1</label>
<input type="checkbox" name="options[]" class="options" id="option2">
<label for="option2">Option 2</label>
```

Puis, en CSS, nous cachons les boutons :

```
input[type="checkbox"].options {
    display:none;
}
```

Pour ensuite travailler l'apparence des <label> :

```
input[type="checkbox"].options + label {
    border: 1px dashed #ccc;
    padding:0.5em;
    cursor:pointer;
}
```

Puis l'apparence des <label> lorsque leur bouton est coché :

```
input[type="checkbox"].options:checked + label {
    border: 1px solid #000;
}
```

Remarque : cette personnalisation de boutons checkbox ouvre la porte à de nombreuses nouvelles fonctionnalités, même hors des formulaires. La capacité des boutons checkbox à conserver un état cliqué et à changer d'état à chaque clic, tels de petits interrupteurs, ne se retrouve pas ailleurs en HTML/CSS. Un exemple parmi d'autres : un bouton « burger » déployant une navigation...

Exemple : animation d'élément au chargement de la page

Selon la nature du site, il peut s'avérer appréciable d'animer le chargement des éléments. Des éléments qui apparaissent avec un effet de fondu ou qui se déplacent vers leur emplacement peuvent donner une impression de vie et de dynamisme aux informations de la page.

La position relative et l'opacité font bon ménage avec la propriété animation pour obtenir ce genre d'effets :

```
@keyframes a1 {  
    from { opacity:0; top:-5em; left:-5em; }  
    to { opacity:1.0; top:0; left:0; }  
}  
h1 {  
    position:relative;  
    opacity:1.0;  
    top:0;  
    left:0;  
    animation-name: a1;  
    animation-duration: 1s;  
}  
  
<h1>Animation</h1>
```



Variante sur base d'une transformation (translation horizontale) :

```
@keyframes a1 {  
    0% { transform: translateX(-100%); }  
    100% { transform: translateX(0); }  
}  
h1 {  
    animation-name: a1;  
    animation-duration: 1s;  
}
```

Exemple : navigation déroulante

Une navigation à plusieurs niveaux, qui se déploie au survol ? C'est réalisable en HTML + CSS ! Pour une telle navigation à plusieurs niveaux, l'HTML est un peu plus chargé que pour une simple navigation, car nous avons besoin d'utiliser des listes imbriquées pour agencer les liens.

```
<nav>
  <ul>
    <li><a href="#">Lien 1</a></li>
    <li><a href="#">Lien 2</a>
      <ul>
        <li><a href="#">Lien 2.1</a></li>
        <li><a href="#">Lien 2.2</a></li>
      </ul>
    </li>
    <li><a href="#">Lien 3</a></li>
    <li><a href="#">Lien 4</a>
      <ul>
        <li><a href="#">Lien 4.1</a></li>
      </ul>
    </li>
  </ul>
</nav>
```

Et le CSS :

```
nav ul {
  list-style:none;
  margin:0;
  padding:0;
  display:flex;
  flex-direction:row;
}
nav ul li {
  position:relative;
  top:0; left:0;
}
nav ul li ul {
  flex-direction:column;
  position:absolute;
  top:100%; left:0; right:-25%;
  display:none;
}
nav ul li:hover>ul {
  display:flex;
}
nav ul li a {
  padding:1em 2em;
  display:block;
  background-color:seagreen;
  color:white;
  text-decoration:none;
}
nav ul li a:hover { background-color:mediumseagreen; }
nav ul li a:active { background-color:lightgreen; }
```

Navigations de 2^e niveau

Les marges et puces par défaut des listes sont annulées.

Lien 1	Lien 2	Lien 3	Lien 4
	Lien 2.1		
	Lien 2.2		

Les sont en position relative pour servir de repère à d'éventuelles sous-navigations, en position absolue.

Pour étendre un peu le 2^e niveau vers la droite, l'écart right est à -25%.

L'astuce du menu déroulant, consiste à cacher la <nav> de 2^e niveau et à l'afficher lors du survol de son parent.

Chapitre 3 : Ergonomie des sites web



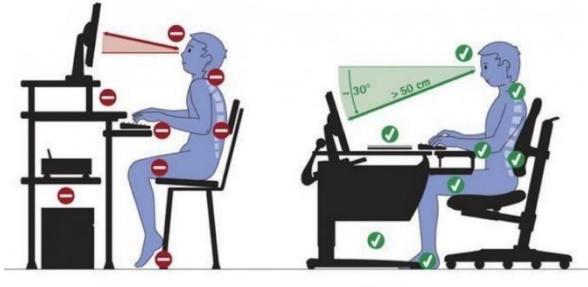
Chapitre 3 : Ergonomie des sites web

3.1 L'ergonomie

L'ergonomie

L'ergonomie est une discipline qui vise à concevoir les outils, les machines et les dispositifs utiles à l'homme de manière à les rendre confortables, sécurisés et efficaces.

Alors que l'ergonomie générale s'intéresse à tous les dispositifs et outils utilisés par l'homme, l'ergonomie web en est un sous-ensemble visant à concevoir des sites web adaptés à l'homme.



Ce chapitre traite des principes fondamentaux de l'ergonomie générale afin de les appliquer à la création de sites web.

L'ergonomie informatique

De nombreuses personnes se plaignent de l'informatique qui est un domaine rigide, ne tenant pas toujours compte des besoins humains.

L'outil (la machine) ne doit pas nous pénaliser mais nous aider. L'ergonomie informatique tente d'inverser ce type de rapports en analysant les besoins des utilisateurs et en adaptant les interfaces informatiques afin de les rendre plus humaines.

Prenons pour exemple une manipulation de base dans Windows : déplacer un fichier dans un dossier. Pour exécuter cette manipulation, de nombreuses possibilités s'offrent à nous : clic-droit > couper / clic droit > coller ; ctrl-x/ctrl-v ; glisser/déposer ; etc. Le fait qu'il y ait autant de possibilités pour réaliser une même manipulation est dû à une démarche ergonomique visant à rendre l'interface de Windows plus humaine : **en proposant différentes façons de réaliser une action, on augmente les chances pour l'utilisateur de trouver une façon qui lui convienne.**



UI & UX design

Dans les années 2010, suite à l'évolution de l'industrie du numérique, de nouveaux domaines d'expertise émergent. L'ancien « expert en ergonomie web » laisse place aux postes d'UI designer et d'UX designer.

L'**UI designer** ou « designer d'interface » se charge de l'ergonomie des visuels, des interfaces.

L'**UX designer** ou « designer d'expérience utilisateur » se préoccupe de tout le parcours de l'utilisateur en se concentrant sur son ressenti émotionnel et ses besoins. Il tente d'améliorer la satisfaction des utilisateurs tout au long de leur parcours.

Conception de site

La conception d'un site web commence par une phase de réflexion et de planification : choisir le sujet, identifier le public visé, déterminer les fonctionnalités et les technologies.

Fonctionnalités – Il est encore un peu tôt pour rêver de fonctionnalités dynamiques. Pour les zones d'administration, les sessions, la vente en ligne, le multi-linguisme, les champs de recherche, les animations, mieux vaut attendre de maîtriser JavaScript, PHP et les bases de données.

Contenus - Il reste néanmoins d'autres critères à déterminer avant de commencer : combien de pages compte le site ? Quel sera leur contenu ? Le site dispose-t-il d'un logo ? D'un titre ? D'un slogan ? De lien vers Facebook/Twitter ?

Ergonomie – N'oubliez pas tous les conseils d'ergonomie vus au cours. Votre site s'adresse-t-il à un public averti (rapidité d'utilisation) ou à un large public (aisance d'utilisation) ? Le contenu est-il fortement textuel (besoin de lisibilité) ou plutôt imagé ?

Navigation – La complexité de la navigation est fortement liée au nombre de pages de votre site : pour un site de 3 pages, une navigation simple fera l'affaire, alors que pour un site de 112 pages, il faudra classer ces pages en groupes et en sous-groupes, prévoir plusieurs blocs de navigation et peut-être des menus déroulants.

Les maquettes : zonings, wireframes, layouts, mockups

La réalisation de la maquette (ou *mockup*) est une étape indispensable à tout bon projet web et a pour objectif de gagner du temps : il est plus rapide de modifier une maquette que le code.

Chacun son style : vous pouvez crayonner sur un bout de papier, griphoner le dos d'un sous-verre ou manipuler un logiciel de dessin.

Les formes de **maquettes** sont nombreuses et ont des degrés de précision variables : *zoning*, *sketch*, *wireframe*, *template*, *layout*, prototype, ... vocabulaire quelque peu variable d'une agence web à l'autre : il est parfois difficile de s'entendre sur les définitions exactes.

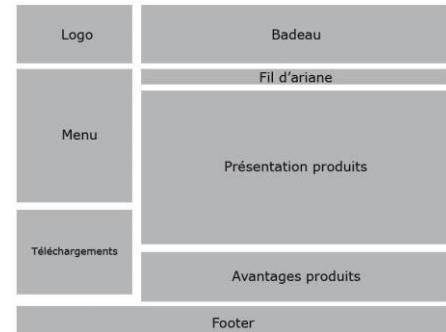


Figure 5 - zoning

Le **zoning** est une représentation grossière des zones du site sous forme de surfaces colorées ou encadrées. Il permet de planifier le partage des zones de la page.

Le **wireframe** est un zoning apportant davantage de précisions : on y voit les contenus (parfois fictifs), les titres, des textes (parfois représentés par des lignes horizontales), les images et vidéos (parfois représentées par un rectangle avec une croix diagonale), les intitulés de la navigation et certaines fonctionnalités comme des formulaires, etc.

Le wireframe peut être réalisé sur papier ou à l'aide d'un logiciel.

Le **layout** est une image réalisée sur ordinateur et présentant le rendu final du site. Il permet d'avoir un aperçu très proche du rendu final du site.

Le **prototype** quant à lui, fait intervenir les langages HTML et CSS afin de permettre quelques fonctionnalités de base comme la navigation.

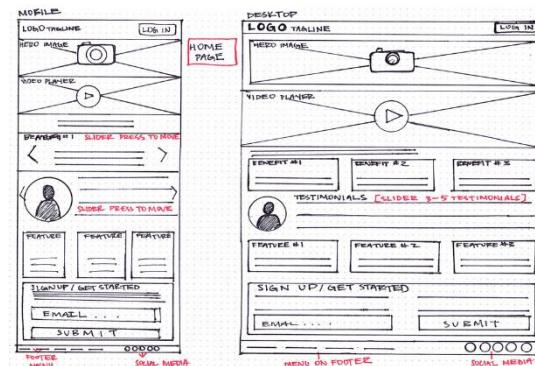


Figure 6 – wireframe à la main

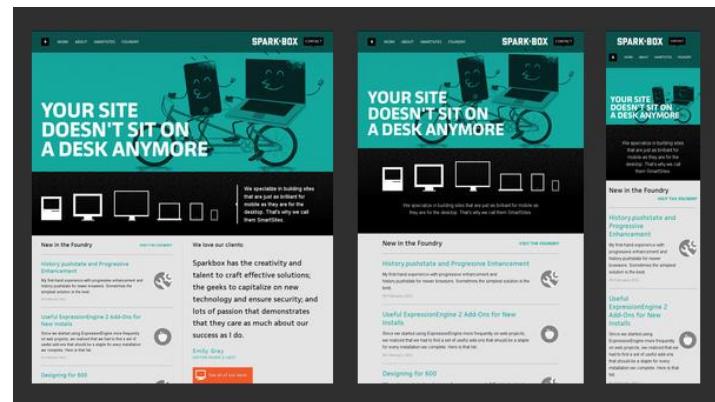
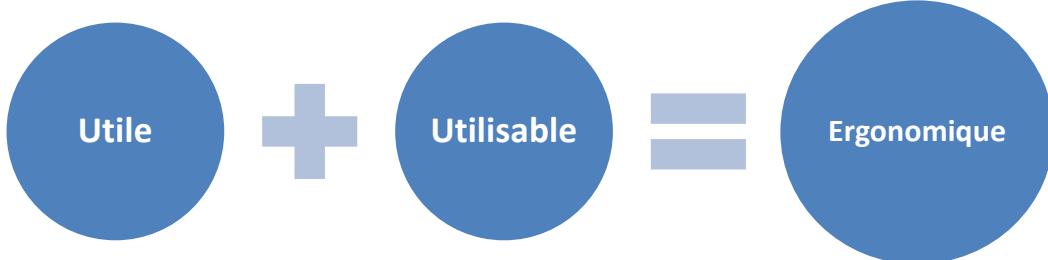


Figure 7 - 3 layouts : desktop, tablette et mobile

Ces 4 formes de maquette sont souvent déclinées pour plusieurs périphériques : desktop et mobile, parfois tablettes.

3.2 Utilité et utilisabilité

Un site web peut être qualifié d'**ergonomique** lorsqu'il répond aux critères d'**utilité** et d'**utilisabilité**.



Utilité

Utilité : disponibilité des informations recherchées par l'internaute.

L'optimisation de l'utilité d'un site commence par les questions suivantes :

- Quel est le sujet du site ?
- Quelles informations cherchent les visiteurs ? Quels sont leurs besoins ?
- Le site contient-il toutes les informations éventuellement recherchées par le visiteur ?

L'utilité concerne le contenu du site et pas son interface.

Attirer les internautes grâce à l'utilité de votre site ne suffit pas. Pour que vos visiteurs restent et profitent de l'utilité de votre site, celui-ci doit être utilisable.

Utilisabilité

Utilisabilité : façon d'accéder aux informations du site.

- L'internaute trouve-t-il ce qu'il cherche aisément ? -> critère d'**efficacité**
- L'internaute trouve-t-il ce qu'il cherche rapidement ? -> critère d'**efficience**
- L'internaute éprouve-t-il du plaisir à naviguer sur notre site ? -> critère de **satisfaction**

L'utilisabilité nécessite l'identification du profil des visiteurs : créons-nous un site **tout public** devant être abordable à des personnes âgées ou à des personnes n'ayant pas l'habitude de surfer sur le Web ? Dans ce cas, nous favoriserons **l'efficacité du site**.

Créons-nous un site pour un **public expert**, habitué au sujet ou au site lui-même ? Par exemple un site de jeux vidéo pour de jeunes entre 18 et 25 ans nés avec un clavier entre les mains ? Ou encore une interface d'administration web qui sera utilisée tellement régulièrement par son administrateur qu'il en connaîtra chaque fonctionnalité ? Dans ce cas, nous favoriserons **l'efficience du site**.

Ou créons-nous un site où les visiteurs ne cherchent pas d'information précise mais surfent pour la détente et le plaisir ? Dans ce dernier cas, nous favoriserons **la satisfaction**.

Ces 3 critères sont présents dans tous les sites, mais **l'identification du public nous permet de déterminer quel critère est à favoriser au détriment des 2 autres**.

Efficacité

Efficacité : l'aisance d'utilisation et la facilité d'apprentissage. L'efficacité doit être privilégiée lorsque le public visé est grand et hétéroclite. Par exemple, pour le site des horaires de bus de la TEC ou le site d'un opérateur de télécommunication.

Lors de **la conception d'un site orienté efficacité**, on considère que les visiteurs se comportent comme s'ils découvraient le site pour la première fois. Il peut d'ailleurs être utile de procéder à quelques jeux de rôles simulant le parcours de visiteurs néophytes (*noobs*) recherchant diverses informations.

La navigation d'un site est parmi les fonctions les plus difficiles à établir en termes d'ergonomie, surtout pour de vastes sites contenant des centaines de pages.

Dans un **site orienté efficacité**, on proposera souvent à l'utilisateur de cliquer plusieurs fois parmi des navigations simplifiées, plutôt que de lui demander de cliquer qu'une ou deux fois parmi de longues listes de liens compliqués. Cette deuxième option sera celle retenue pour les **sites orientés efficience**, où la priorité est donnée à la rapidité et non à la facilité d'utilisation.



Figure 8 - Le site de Proximus : un bel exemple de site orienté efficacité !

- Le nombre d'éléments dans les navigations est réduit : 3 à 6 liens par bloc de navigation.
- La navigation « commerciale » (en bas) comporte des intitulés très courts : « Packs », « Mobile », etc.
- Cette même navigation est renforcée d'icônes : même plus besoin de lire les textes... La combinaison icône + texte court est la championne de l'efficacité !
- Les liens réagissent visuellement au survol et au clic.
- Les zones cliquables sont larges : on peut cliquer légèrement à côté des textes ou des icônes.
- Un fil d'Ariane est présent sous le logo Proximus, discret mais utile pour s'y retrouver dans des sites aussi vastes.
- Un champ de recherche pour aider les visiteurs un peu perdus.
- Une couleur spécifique (rose), avec un fort contraste avec les autres couleurs du site, est réservée à l'élément le plus important de la page : la promotion du moment !
- Le site contient peu d'informations textuelles pour permettre à l'utilisateur de se concentrer sur son choix de navigation.

Efficience

Efficience : la rapidité d'utilisation. L'utilisateur doit pouvoir accomplir ses objectifs rapidement et avec le moins d'erreurs possible. L'efficience est importante dans les applications métiers et les interfaces spécialisées. Par exemple, le site jeuxvideos.com qui s'adresse à un public averti et habitué au Web.

Lors de la conception d'un site orienté efficience, on fera attention au temps lié à chaque action : éviter les rechargements de pages intempestifs ; proposer de nombreuses fonctionnalités sur une même page ou dans un même menu ; utiliser des menus déroulants pour caser un maximum de liens sur un minimum de place ; rapprocher (en distance et en nombre de clics) les fonctionnalités les plus souvent utilisées et éloigner les autres.

En matière de logiciels, nous pourrions comparer Paint à Photoshop : Paint propose un nombre restreint de fonctionnalités représentées par de grosses icônes, alors que Photoshop comporte des miriades de fonctionnalités représentées par des intitulés plus ou moins longs et parfois dissimulés dans des sous-sous-menus.

Souvent, les sites font le choix de combiner l'efficacité et l'efficience, en appliquant l'efficacité aux parties les plus visibles du site comme la navigation primaire ou la page d'accueil.

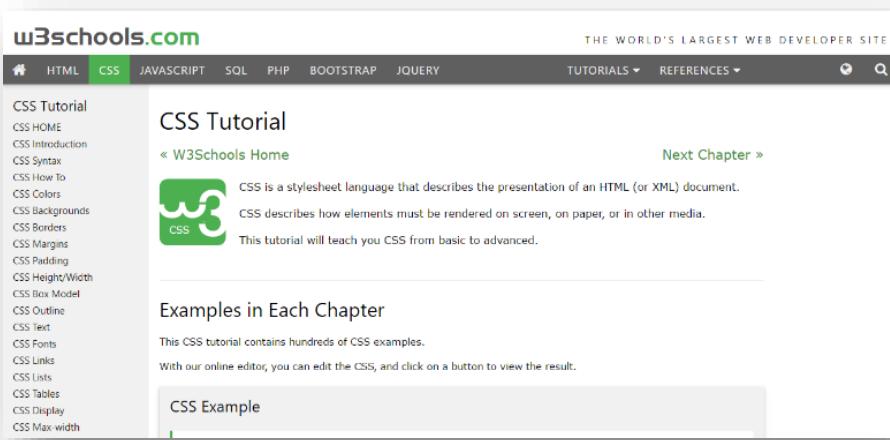


Figure 9 - Le site de W3Schools : comme sur de nombreux sites orientés **efficience**, on retrouve une navigation de 1^{er} niveau orientée **efficacité** (seulement 7 liens, intitulés courts, utilisation de quelques icônes, zones cliquables étendues, etc.), cependant, plus on naviguera dans le site, plus on ressentira qu'il est destiné à un public averti. La navigation secondaire (marge gauche) nous le démontre :

- Moins d'images, d'icônes et de couleurs
- Intitulés de liens plus longs
- Zones cliquables plus petites
- Nombre de liens beaucoup plus élevé

Satisfaction

Satisfaction : c'est ici que l'ergonomie web se détache de l'ergonomie informatique et de l'ergonomie en général : il est tout à fait envisageable de visiter un site web pour se détendre, se promener. C'est toutefois plus difficilement envisageable avec un traitement de texte, un marteau-piqueur ou un distributeur de boissons.

Ce critère devra surtout être privilégié sur des sites où le visiteur ne recherche pas une information précise.

Ces **3 critères d'utilisabilité** cohabitent. Il n'est pas question de n'en travailler qu'un seul et d'oublier complètement les autres. Il est surtout question d'identifier celui qui doit être favorisé.

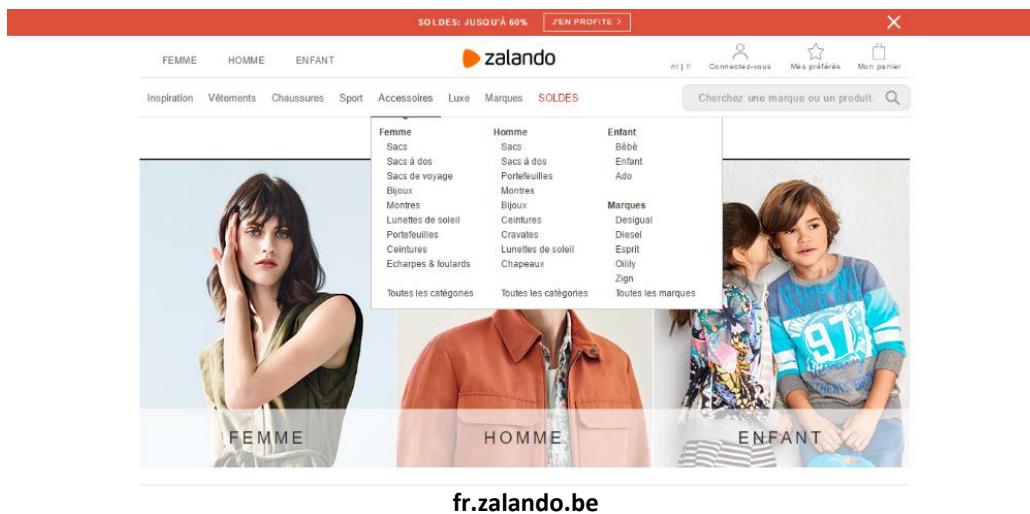
Exercice : site efficace ou efficient ? Pourquoi ? Citez 3 indices !

The screenshot shows a dark-themed version of the PHP documentation website. At the top, there's a navigation bar with tabs for 'php', 'Downloads', 'Documentation' (which is currently selected), 'Get Involved', and 'Help'. A search bar is located at the top right. The main content area has a header 'Référence des fonctions'. Below it, a blue box contains the text 'Astuce Voir aussi Catégorie/Liste des extensions.' followed by a bulleted list of PHP extension names: APC, APCu, APD, bcompiler, BLENC, Gestion des erreurs, htscanner, inclued, Memtrack, OPcache, Contrôle de l'affichage, Options PHP et informations PHP, and runkit. To the right of the main content, there's a sidebar titled 'Manuel PHP' with links to various sections like Copyright, Manuel PHP, Au moment de commencer, Installation et configuration, Référence du langage, Sécurité, Caractéristiques, Référence des fonctions (which is bolded to indicate the current section), Dans le cœur de PHP : Guide du Hacker, FAQ, and Annexes. The footer of the page features the 'php.net' logo.

Exercice : site efficace ou efficient ? Pourquoi ? Citez 3 indices !



Exercice : site efficace ou efficient ? Pourquoi ? Citez 3 indices !



3.3 Quelques (mauvaises) idées reçues sur l'ergonomie web

La sainte règle des 3 clics

- Bien que le nombre de clics soit une information de distance sur le Web, la difficulté pour atteindre une information ne dépend pas que de ce nombre.
- La complexité mentale (comparaison, choix, raisonnement) cachée derrière chaque clic est un critère d'autant plus important.
- Les visiteurs ne partent pas parce qu'ils ont cliqué trop de fois, ils partent parce qu'ils ont l'impression de ne pas être sur la bonne voie.
- 2 clics simples valent mieux qu'un clic compliqué.
- Attention, cette règle repose tout de même sur un bon fond. Gardez simplement à l'esprit qu'elle est incomplète.



Les internautes sont des idiots

Considérer les internautes comme des idiots peut sembler un bon conseil pour s'assurer qu'un maximum de visiteurs s'y retrouvent. Il ne faut pour autant pas assister les internautes à l'extrême, au détriment d'autres aspects du site.

La conception d'un site demande de deviner le comportement des cohortes de visiteurs. Trois erreurs sont alors fréquentes :

- Sous-estimer les internautes : attention de ne pas simplifier à l'extrême (efficacité) au détriment de la rapidité (efficience).
- Surestimer les internautes : « tout le monde sait que pour retourner en page d'accueil, il faut cliquer sur le logo... » Les internautes ne sont pas tous des informaticiens et nos connaissances ne sont pas forcément les leurs.
- Se projeter : « si moi j'y arrive, les autres n'ont qu'à y arriver aussi ».



Encore une fois, l'important est de connaître le public ciblé.

Les internautes lisent en « F »

Si la lecture d'un livre se fait en « F », cela ne s'applique que très peu aux pages web qui sont plus souvent parcourues en vitesse que lues en profondeur. Lors de ce balayage rapide de la page par le regard du visiteur, ce sont les éléments les plus « criants » qui vont attirer son attention : les images, les titres, les textes de grande taille, les couleurs vives, les majuscules, les mouvements, etc.



Il est important de maîtriser ces aspects « criants » pour guider le visiteur. En appliquant les bonnes tailles de texte et les bons contrastes, on peut lui faire comprendre toute la hiérarchie de titres d'une page en un clin d'œil. En appliquant une couleur vive à un élément, on peut attirer son regard sur une annonce ou une publicité.

Attention, ces éléments « criants » peuvent provoquer l'effet inverse s'ils sont mal utilisés : par exemple une animation en boucle qui déconcentrerait le visiteur ou des tailles de textes/titres qui ne seraient pas proportionnelles à leur importance.

Agencer une page web en « F » n'aurait de sens que pour des informations fortement textuelles, mais pas pour des supports destinés à être parcourus rapidement.

Les internautes ne scrollent pas

Trop souvent, on croit que les internautes ne scrollent pas. Or on n'en sait rien. Tout dépend du contexte. Si les internautes ont l'impression que le site correspond à leurs besoins et que leur objectif est contenu dans la suite de la page, ils scrollent.

Aux débuts du Web, de nombreux internautes ignoraient qu'il était possible de descendre dans la page. Il fallait réellement en tenir compte lors de la conception des sites web. Aujourd'hui, et en partie grâce à la démocratisation de la molette sur les souris, ce critère a perdu son importance. La tendance se serait même inversée : les sites actuels nécessitent souvent trop de scroll en écartant trop les informations.

Lorsqu'il s'agit d'une page de contenu, il est tout à fait légitime que la page soit scrollable. Lorsqu'il s'agit d'une page navigable ou de transaction, mieux vaut que les contenus situés sous le seuil de scroll soient secondaires.

On le fera en Web 2.0

On pense souvent que le recours à des concepts ou des technologies données a une influence sur la qualité ergonomique des sites web. Cette démarche ne s'avère pas toujours fausse, mais reste insuffisante.

Le design : ennemi juré de l'ergonomie

D'un côté « l'ergonomie tue le design », de l'autre « le design ruine la qualité ergonomique à laquelle on aurait pu prétendre » sont des idées couramment reçues sur les rapports entre ergonomie et design.

Il est du ressort d'une décision stratégique de choisir entre une interface extrêmement facile à utiliser ou une interface extrêmement graphique reflétant une image de marque. Ce n'est cependant pas pour autant que l'ergonomie et le design sont entièrement antinomiques. Chacun apporte au site son lot d'avantages et bien qu'il faille parfois prioriser l'un au détriment de l'autre, les deux sont toujours présents.

Pour l'ergonomie, on verra à la fin

L'ergonomie n'est pas une composante indépendante des autres métiers, ni une discipline ponctuelle, mais une préoccupation générale qui doit être présente tout au long d'un projet.



Exemples de retour sur investissement de l'ergonomie web

- Refonte des sites d'IBM en 1999 : une semaine après, les ventes augmentaient de 400%
- Refonte d'un site de vente de cheese-cakes : augmentation de 900%
- Refonte du site de Bell Canada : 450000 visiteurs par semaine au lieu de 300000.
- Refonte de la page d'accueil du site de l'artiste Richard Scott : interface proposant plusieurs peintures sous formes de vignettes plutôt qu'une seule. Augmentation des ventes de 30%.
- Etc.

3.4 Lois de Gestalt

Le **gestaltisme**, ou psychologie de la forme, définit nos capacités à reconnaître des images. Spontanément, notre perception et notre représentation mentale traitent des groupes d'éléments comme des ensembles structurés et non comme une simple juxtaposition de ceux-ci.

Nous sommes tous dotés d'une formidable capacité à instinctivement reformer, compléter, rassembler et identifier des éléments individuels en des groupes sensés.

L'ensemble d'étoiles devient une constellation... Le nuage nous rappelle vaguement un animal... Quelques traits deviennent un visage. Le tout prend un sens et devient plus que la somme des parties.



Wenceslas Hollar - Landscape shaped like a face

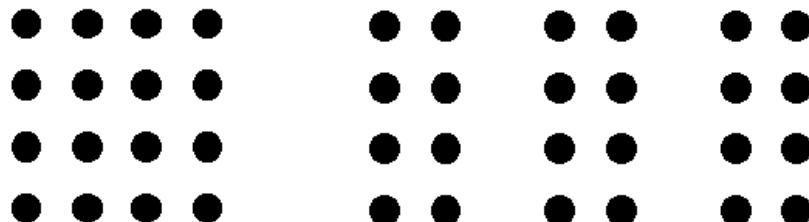
Le **gestaltisme** dissèque ce phénomène et énumère les lois principales suivantes :

- **La loi de la bonne forme** : loi principale dont les autres découlent : tout ensemble informe tend à être perçu comme une forme simple. Par exemple un ensemble de points qui évoquerait un carré ou un cercle.
- **La loi de continuité** : tendance à percevoir des points rapprochés comme des lignes continues.
- **La loi de proximité** : tendance à regrouper des éléments selon leur proximité.
- **La loi de similitude (ou similarité)** : tendance à regrouper les éléments selon des critères visuels communs : couleur, forme, taille, orientation ou vitesse de déplacement.
- **La loi de familiarité** : tendance à reconnaître prioritairement les formes les plus familières et les plus significatives.

Nous allons maintenant étudier certaines de ces lois qui peuvent s'appliquer aisément à la conception de nos sites web.

La proximité

La loi de proximité énonce la tendance de notre cerveau à regrouper les éléments rapprochés. Autrement dit, on considère que deux éléments qui sont proches physiquement entretiennent un rapport.



Normalement, votre cerveau a été capable de rapidement identifier 2 groupes d'éléments dont un groupe (celui de droite) constitué de 3 sous-groupes.

Comment mettre cette loi à profit dans nos sites web ? D'une part, nous allons rapprocher les éléments qui entretiennent un rapport fonctionnel. D'autre part, nous allons éloigner les éléments qui n'ont rien en commun.

À la lecture de cette loi, elle semble évidente. Pourtant, il est fréquent de rencontrer des sites oubliant de l'appliquer.

Exemple d'erreur de proximité dans un formulaire :

Pseudo :

Mot de passe :

Commentaire :



Pseudo :

Mot de passe :

Commentaire :



→ Cet exemple fait défaut à la loi de proximité : les champs sont plus rapprochés les uns des autres que les intitulés des champs. **Le rapport fonctionnel intitulé/champ est brisé.**

→ Dans cet exemple, les intitulés sont tous légèrement plus proches de leur champ que des autres éléments. Cet exemple assure une meilleure lisibilité et une meilleure compréhension par le visiteur.

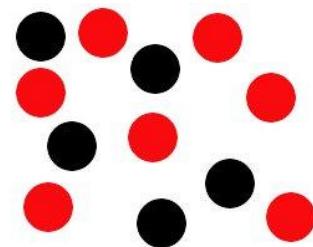
La familiarité

La loi de familiarité énonce que les formes familières sont perçues comme plus significatives.

La similarité

La loi de similarité ou de similitude énonce la tendance de notre cerveau à regrouper les éléments qui se ressemblent par leur forme, leur taille, leur couleur, leur orientation ou leur vitesse de déplacement.

Les ressemblances ou différences sont perçues comme le signe que les objets sont comparables ou opposables.



Comment mettre cette loi à profit dans nos sites web ? L'application de cette loi peut améliorer l'utilisabilité d'un site. Par exemple en appliquant une couleur commune à des fonctionnalités communes. Ce genre de détails rend l'interface plus intuitive et facilite son utilisation.

Cette similarité se retrouve également dans des listes d'éléments. Toute série ou liste d'éléments se doit de comporter des critères visuels communs par leur taille, leur forme, leur orientation, etc. même si l'on peut se permettre de les faire varier quelque peu.

Ainsi, les intitulés de navigation sont également concernés dans leur forme grammaticale :

- [Page d'accueil](#)
- [La Boutique](#)
- [Visiter la galerie](#)
- [Contactez-nous](#)



- [Accueil](#)
- [Boutique](#)
- [Galerie](#)
- [Contact](#)



→ Dans cet exemple, la liste comporte des intitulés de formes grammaticales variables : nom commun, nom commun accompagné d'un article, verbe à l'infinitif, verbe à l'impératif...

→ Cette liste est plus agréable à la lecture grâce à sa forme plus courte mais surtout grâce à sa **conformité grammaticale**.

3.5 Autres principes fondamentaux de l'ergonomie

La loi de Fitts

Le temps nécessaire à atteindre une cible est proportionnel à la distance à laquelle elle se trouve et inversement proportionnel à sa taille. C'est-à-dire qu'une cible est d'autant plus facile à atteindre qu'elle est proche et grande.

On appelle cela le **Fittsizing** : choisir la taille des éléments suivant leur importance et compenser leur éloignement par la taille.

Comment mettre cette loi à profit dans nos sites web ? En augmentant la taille des éléments cliquables, nous diminuons également le nombre de clics « ratés », c'est-à-dire le nombre de fois que le visiteur rate la cible et doit recliquer. Il faut savoir cependant que ce problème n'est pas majeur. En effet, les internautes sont tellement habitués à corriger leurs erreurs de clic que cela devient un réflexe.



La figure ci-dessus représente une image générée par l'outil ClicDensity et qui permet d'observer les endroits les plus cliqués par les utilisateurs. Il est intéressant de constater le nombre d'erreurs de clics, surtout dans le pied de page où les éléments sont plus petits.



Dans cet exemple, c'est la troisième phrase qui est la plus simple à cliquer :

- Pour consulter l'article, cliquez [ici](#)
- Pour consulter l'article, [cliquez ici](#)
- [Pour consulter l'article, cliquez ici](#)



Dans le même ordre d'idée, il faudra **toujours s'assurer que les surfaces des boutons soient entièrement cliquables** et pas seulement le texte ou l'icône qu'ils contiennent. S'il ne s'agit pas d'un bouton, mais d'un texte cliquable, il est conseillé d'utiliser l'espace blanc autour. Techniquelement, cela revient souvent à travailler les éléments <a> avec *display:block* ; et ajouter une marge intérieure avec *padding*.



Dans les formulaires, il est recommandé de pouvoir aussi bien sur le libellé des champs que sur les champs eux-mêmes. Techniquelement, cela revient à utiliser l'élément <label for="">.



Concernant la **distance**, il est difficile de prédire où se trouve le pointeur de la souris de votre visiteur. Les endroits les plus faciles d'accès sont les 4 coins de l'écran et le centre.



La distance est également à optimiser en termes de **regard du visiteur**. Par exemple, pour les formulaires, il est préférable de placer le bouton de validation juste après le dernier champ puisque c'est l'endroit où se trouvera le regard du visiteur après avoir entré les données.



Enfin, même si c'est plus appliqué aux logiciels et aux applications qu'aux sites web, les raccourcis claviers permettent de simplifier les actions. C'est ce que l'on appelle la **proximité immédiate**. Il en est de même pour le clic droit qui laisse apparaître un menu où que l'on clique dans la fenêtre.

L'affordance



Les **affordances** sont les **possibilités d'action suggérées** par les caractéristiques d'un objet. Par exemple, face à une porte, une série d'éléments vous permettent de deviner l'action requise pour son ouverture : pousser, tirer, faire coulisser ? Parmi ces indices se trouvent des **signes implicites** : la forme de la poignée, son emplacement, une barre horizontale du type sortie de secours, des gonds visibles, etc. et des **signes explicites** : un écrit au « poussez » ou « tirez », etc.

Lorsque vous arrivez sur un site web, des indices visuels vous permettent de déterminer quels éléments sont cliquables ou survolables. Si le visiteur hésite et teste vos éléments en les survolant ou en les cliquant, c'est que vos signes d'affordances sont insuffisants.

Comment mettre cette loi à profit dans nos sites web ? À la fois en améliorant l'affordance des éléments permettant des actions et à la fois en évitant les fausses affordances :

- Ne soulignez pas les textes non cliquables (sauf pour indiquer une partie de texte corrigée) !
- Évitez de donner de fausses apparences de bouton aux éléments de la page.
- Appliquez le curseur en forme de main (`cursor:pointer;`) aux éléments cliquables. Notez que le navigateur s'en charge déjà pour les éléments `<a>`, mais pas pour les éléments rendus cliquables par l'ajout de JavaScript.
- Exploitez les autres curseurs indiquant des possibilités d'étirement, de déplacement, etc.
- Utilisez une couleur de texte propre aux liens.
- Exploitez les sélecteurs de pseudo-classes `:hover`, `:focus` et `:active` pour tous les éléments cliquables de votre site afin d'inviter visuellement à une action.
- Évitez d'appliquer des changements visuels au survol sur des éléments non cliquables.

Le nombre magique de Miller

À moins d'être des super-héros, les internautes sont constitués comme la plupart des êtres humains, c'est-à-dire avec des capacités mentales limitées.

Miller est un psychologue qui recense dans les années 50 tout un ensemble d'expériences et de preuves scientifiques appuyant le constat suivant : **au-delà de 7 objets, la mémoire s'embrouille**. Attention, nous parlons ici de la **mémoire de travail ou mémoire à court terme**. Cela signifie que lorsque qu'un être humain se retrouve dans une situation nouvelle avec plus de 7 éléments à visualiser, analyser et comparer, il éprouve des difficultés de mémorisation qui le pousse à visualiser et analyser plusieurs fois chaque élément.

La mémoire de travail peut être vue comme la structure mentale permettant de stocker des informations temporaires, un peu comme la RAM d'un PC.

Ce nombre de « 7 » est approximatif : il s'agit plutôt d'un intervalle allant de 5 à 9, suivant les personnes et les situations.

Comment mettre cette loi à profit dans nos sites web ? Cette limite cérébrale est surtout à prendre en compte dans la conception des navigations :

- Plus le nombre de choix de navigation sera limité, plus son utilisation sera facile.
- Ne pas dépasser 7 choix par navigation, sauf si vous estimez que le choix est simple (par exemple lorsqu'on demande son pays d'origine à un visiteur) ou si vous vous adressez à un public expert (voir chapitre sur l'efficience).
- Si les choix sont trop nombreux, regroupez-les en catégories et proposez plusieurs choix consécutifs à l'utilisateur : ainsi, 24 choix peuvent devenir 4 fois 6 choix. Ce qui est plus efficace.

La lisibilité

La lisibilité du texte dépend de nombreux facteurs : la couleur du texte, la couleur de fond, le contraste entre les deux, la police, la taille du texte, la graisse du texte, etc.

Attention ! Écrire en majuscules, en souligné ou en italique, nuit à la lisibilité des textes. Ce ne sont pas des pratiques proscribes, mais il est tout de même déconseillé de les appliquer sans raison à de longs textes.

Gardez à l'esprit que sur le Web, les visiteurs sont pressés. Il est rare qu'un visiteur lise entièrement un texte. Évitez donc les textes trop longs et les paragraphes trop longs. Évitez les textes écrits sur toute la largeur d'une page et préférez travailler plusieurs colonnes de textes.

Enfin, utilisez l'élément **** pour mettre en évidence quelques mots clés ou extraits de textes accrocheurs.

Chapitre 4 :

Référencement web



Chapitre 4 : Référencement web

Les débuts du référencement

Au début du Web, dans les années 1990, le nombre de sites croît rapidement. Deux outils de recherche d'informations font alors leur apparition : l'annuaire et le moteur.

Les annuaires (comme Nomade, Yahoo! ou Dmoz), très en vogue dans les années 90, permettaient de rechercher un site en parcourant des catégories et sous-catégories de sujets. Cette méthode de recherche finit cependant par disparaître au profit des moteurs dont le fonctionnement plus rapide sût conquérir le public.

Les moteurs de recherche de l'époque (comme Lycos ou Altavista) optaient pour un fonctionnement plutôt primaire : pour chaque page, il totalisait le nombre d'occurrences du ou des mot-clés cherchés. Le premier résultat était la page en comportant le plus... de quoi ouvrir la porte à certaines dérives !

Face à la croissance spectaculaire du Web et à la concurrence acharnée des référencEURS, les moteurs durent évidemment s'adapter. Aujourd'hui, Google et Bing utilisent des algorithmes (secrets) à la complexité extraordinaire : à base d'intelligence artificielle, de réseaux de neurones et d'apprentissage automatique. Les mises à jour des algorithmes sont fréquentes : tantôt pour bloquer une faille exploitée par des référencEURS, tantôt pour promouvoir les démarches d'accessibilité, tantôt pour s'adapter à de nouvelles technologies, etc.

Les algorithmes de référencement

C'est le très sérieux New York Times qui, à l'occasion d'une immersion au sein des bureaux de Google aux USA a pu dévoiler quelques informations sur la manière de travailler du moteur de recherche. Il en est ressorti entre autres que l'équipe chargée de la qualité des recherches modifie l'algorithme secret de Google en moyenne 6 fois par semaine, pour des changements mineurs mais aussi majeurs.

Un exemple de recherche non pertinente est par exemple la requête « French Revolution » qui, lors des élections présidentielles françaises, rentrait principalement des sites Internet commentant l'actualité politique liée aux élections au lieu des événements historiques. La raison était que l'algorithme donnait plus de poids aux pages contenant les deux mots côté à côté qu'aux pages contenant les deux mots séparés. C'est ce genre de situation qui fait que l'algorithme est retouché de manière quasi quotidienne. Les équipes de Google sont donc constamment à la recherche de l'algorithme le plus à même de répondre aux attentes des visiteurs.

C'est sans doute aussi ce qui explique que le métier de référencEUR soit si passionnant.

Le référencement et le positionnement

Il est habituel d'utiliser à tort le terme « référencement » : parfois pour désigner le positionnement d'un site, d'autres fois pour désigner toute la démarche d'optimisation de la visibilité d'un site. Il convient donc de rappeler la définition de ce terme : le référencement est l'indexation d'un site web par le moteur de recherche. C'est-à-dire la simple prise de connaissance de l'existance de ce site.

Le « positionnement », quant à lui, intervient après l'indexation et désigne la démarche visant à améliorer la position d'un site dans les résultats des moteurs de recherche.

La page de résultats (SERP)

La page de résultats d'un moteur de recherche est appelée SERP (*Search Engine Result Page*). En plus des 10 liens naturels classiques, selon la recherche, la SERP peut proposer des liens sponsorisés, un « pack local » si la recherche est géolocalisée, un « knowledge graph » si la recherche est assimilée à une demande encyclopédique, ou d'autres contenus en « position zéro » c'est-à-dire audessus du 1^{er} résultat naturel comme des réponses directes « answer boxes ».

Ligne Claire
<https://www.ligne-claire.be> •
 Ligne Claire organise des événements toutes les semaines de l'année ! Yu-Gi-Oh!, Dragon Ball, Magic, ... LIGNE CLAIRE Grand'Rue 66 7000 Mons Belgique
 Jeux de Cartes à Collectionner Magic The Gathering Pokémon Panini Marvel

Ligne Claire - Accueil | Facebook
<https://fr-fr.facebook.com/Lieux/Mons/Magasin-de-bandes-dessinées> •
 ★★★★ Note : 4,8 - 75 votes
 Ligne Claire - Grand'Rue 66, 7000 Mons - Note de 4,8 sur la base de 75 avis «Je viens de recevoir ma Funko POP Remus Lupin as Werewolf que j'ai ...»

Ligne Claire (Mons) : 2019 Ce qu'il faut savoir pour votre visite ...
<https://fr.tripadvisor.be/.../Mons-Mons-toutes-les-activites> •
 ★★★★ Note : 4,5 - 10 avis
 Ligne Claire c'est votre librairie spécialiste en bd, comics, mangas à Mons en Belgique. Mais aussi en produits TINTIN et MOULINSART dans le Hainaut. Ligne ...

Ligne Claire - Entreprise - Mons | visitMons - Portail ...
www.visitmons.be/.../Shopping/Ligne-Claire-Entreprise-Mons •
 La boutique Ligne Claire est labellisée Mons ville du Cadeau Original. Ligne Claire c'est votre librairie spécialiste en BD, comics, mangas à Mons en Belgique.

Ligne Claire Librairie BD Mons

Site Web Itinéraire Enregistrer
 4,7 ★★★★★ 346 avis Google
 Librairie de bandes dessinées à Mons

Adresse : Grand'Rue 66, 7000 Mons
 Horaires : Ouvert - Ferme à 18:30
 Téléphone : 065 33 48 38
 Suggérer une modification

Événements à venir
 mer. 18 déc. Dragon Ball Card Game - Casual tous les ... 14:00

Figure 8 : pack local (à droite)

Vidéos

- Baratte à beurre 1,6 litres Tom Press pour fabriquer du beurre ... Tom Press YouTube - 14 févr. 2013
- Test d'une baratte à beurre "La bonne graine" Climaxe FR YouTube - 7 mars 2016
- Echiré : un beurre baratté à l'ancienne La Quotidienne YouTube - 2 mars 2016

Résultats Shopping Annonce sponsorisée

baratte manuelle 1,6 litres Brouwland Par Google	Kilner Baratte 35,99 € piccantino.be Par smec	Hofmeister Holzwaren ... 73,00 € Amazon.fr Par Google
--	---	---

Baratte

La baratte est un outil qui permet de transformer la crème de lait en beurre. Le barattage consiste à séparer par un mouvement mécanique les particules de ...

baratte à beurre - Amazon.fr
<https://www.amazon.fr/baratte-a-beurre-k-baratte-a-beurre> •
 Amazon.fr : baratte à beurre ... Kilner Baratte à beurre en verre ... LA BONNE GRAINE - Baratte à Beurre pour Faire son Beurre Maison - Kit Complet Bocal + ...

Figure 9 : vidéos en position zéro, liens sponsorisés et informations encyclopédiques

Le référencement naturel (SEO) et sponsorisé (SEA)

Les résultats des moteurs de recherche se divisent en 2 catégories : les résultats naturels (ou organiques) et les résultats sponsorisés. Les résultats naturels proviennent d'une démarche exempte de processus financier ou publicitaire, appelée référencement naturel, ou en anglais SEO (*Search Engine Optimization*). À l'opposé, les résultats sponsorisés sont issus du SEA (*Search Engine Advertising*). L'association SEO et SEA se nomme SEM (*Search Engine Marketing*).

Une campagne **SEA** passe le plus souvent par l'outil **Google AdWords** permettant de réserver des mots-clés à des tarifs variables appelés **CPC** (Coût par Clic) : la somme à payer par clic sur le lien sponsorisé.

Mais le CPC peut aussi être en votre faveur si vous passez par **Google AdSense** pour permettre l'affichage de publicités sur votre site afin de le rentabiliser.

Le SEA peut influencer le SEO depuis que Google prend en compte le trafic des sites dans son algorithme de *ranking* : une campagne SEA ramenant des visiteurs, peut donc améliorer la position d'un site si elle ramène suffisamment de visiteurs.

Bien que les liens sponsorisés soient placés avantageusement dans la page de résultats, les visiteurs ne sont pas dupes et le 1^{er} résultat naturel reste largement le plus cliqué de tous (+ de 50% des clics).

À propos de Google

Google est une société fondée en 1998 par deux étudiants californiens créateurs du moteur de recherche Google. En plus de sa situation économique très confortable, Google est détenteur de plusieurs records parmi lesquels le plus grand parc de serveurs au monde qui compte 2 millions de machines stockant la quasi-totalité du Web, et le nom d'entreprise le plus célèbre au monde devant Microsoft et Coca-Cola.



Le nom de « Google » est inspiré du terme mathématique « gogol » qui désigne le nombre 1^{100} .

En plus du moteur de recherches, Google étend aujourd'hui ses activités à la cartographie numérique avec *Google Maps* et *Google Earth*, à la géolocalisation par satellite, aux outils de traduction en ligne, aux systèmes de messageries instantanées avec *GTalk* (qui n'a pas su s'imposer face à *MSN* et *Skype*) mais aussi avec *Google Wave* (qui tente d'associer messagerie, emails et réseaux sociaux), aux systèmes de lecture des flux RSS avec *Google Reader*, aux services d'analyse de trafic des sites web avec *Google Analytics*, aux finances en ligne avec *Google Finance*, aux agendas en ligne avec *Google Agenda*, au Cloud Computing avec *Google Apps*, aux traitements de texte en ligne par le rachat de l'entreprise *Writely* pour créer *Google Docs*, aux vidéos en ligne par le rachat de *Youtube*, aux navigateurs web avec le lancement de *Google Chrome*, etc.

Punitions Google

Il est déconseillé de tricher avec les moteurs de recherche en profitant de failles dans leur fonctionnement, et ce pour 2 raisons : tout d'abord, Google corrige sans cesse ses failles ce qui fait que le bénéfice tiré ne sera sûrement que temporaire. Ensuite, parce que Google peut sévir.

La punition de premier niveau appelée **sandBox** (ou « bac à sable ») consiste en une désindexation temporaire d'un site. Cette punition est toutefois de moins en moins appliquée par Google.

Une autre punition appliquée par Google est le **déclassement temporaire** : 30 ou 60 places de pénalité de positionnement.

La punition de second niveau, bien plus sévère, est la **blackList** (ou « liste noire ») : une désindexation définitive.

Mots-clés et spiders

Les moteurs de recherche sondent sans cesse la toile en envoyant des **robots** appelés **spiders** ou **crawlers** visiter des sites et recueillir de nombreuses informations, comme les mots-clés. Ces informations sont ensuite enregistrées dans un registre.

À partir des textes recueillis, les moteurs de recherche établissent des liens entre les mots selon qu'ils sont souvent voisins ou pas. Ce travail constitue une immense carte de mots-clés évolutive.

La première étape visant à améliorer le positionnement d'une page, consiste à identifier et placer les mots-clés en espérant constituer une carte proche de celle du moteur de recherche.

Une question revenant fréquemment sur l'écriture de ces mots-clés est : faut-il les dériver ? Par exemple, pour un site parlant d'étudiants, a-t-on intérêt à dériver ce mot « étudiant » au féminin, au pluriel, etc. ? à l'écrire sans accent ? en majuscules ? avec des fautes de frappe ou d'orthographe ? « Étudiante, étudiants, étudier, études, etudiant, ETUDIANT, etudant, ethudiend... » La liste s'avère longue ! De plus, il semble difficile de placer des fautes de frappe et d'orthographe sans « salir » votre site.

La réponse à cette question est : oui, il y a un certain intérêt à dériver vos mots-clés si vos visiteurs les dérivent également.

Heureusement, le moteur de recherche est intelligent et capable de deviner l'objet d'une recherche mal orthographiée : « gooogel », « rayanair » ou « Olivier Andrieux » donnent les résultats souhaités alors que ces orthographies ne figurent pas dans le contenu visé. Ceci sans doute en partie grâce aux nombreux *backlinks* spontanés contenant ces fautes d'orthographe...

Liens internes et liens entrants

Un bon référencement passe forcément par une optimisation de vos liens : qu'ils soient internes (de votre site vers votre site) ou entrants (d'un site externe vers votre site).

Les liens internes permettent aux robots de parcourir l'ensemble de votre site et donc d'indexer toutes vos pages. Lorsque vous codez vos liens internes, veillez à utiliser une syntaxe HTML propre et évitez les liens JavaScript ou d'autres technologies. Quant à l'intitulé du lien, il est essentiel au référencement de vos pages. Placez-y vos mots-clés et évitez les liens dénués de mots-clés comme « Cliquez ici » ou « Découvrez notre offre ».

Quant aux liens entrants (ou *backlinks*), ils sont tout aussi importants mais plus compliqués à obtenir puisqu'ils proviennent d'autres domaines que votre site. Retenez tout de même que rien ne vaut des *backlinks* spontanés, publiés naturellement par des personnes intéressées par les contenus de votre site.

Attaques SEO

Une attaque SEO de type **Google Bombing** est une attaque visant la carte de mots-clés de Google. De très nombreux postes font coïncider des termes qui ne l'étaient pas auparavant, si bien que Google finit par les considérer comme termes voisins ou synonymes. De nombreuses personnalités politiques en ont fait les frais : Georges Bush lié à « *miserable failure* », Nicolas Sarkozy à « trou du cul du Web » et François Hollande à « incapable de gouverner ». Cependant, en 2007 Google a travaillé sur le blocage de ces attaques, les rendant de plus en plus rares. Ce qui n'a tout de même pas empêché Donald Trump d'être assimilé au terme « *idiot* ».

Face à une faille de plus en plus exploitée consistant à publier massivement et artificiellement des *backlinks* (liens entrants vers votre site), la mise à jour « Google Penguin » sort en avril 2012 et applique des malus de positionnement aux sites exploitant cette faille. Les référenciers crient alors au scandale et craignent de leurs concurrents des attaques de type « **Negative SEO** » (ou NSEO) c'est-à-dire des *backlinks* provoquant ce malus.

Duplicate Content

Google est capable de reconnaître les contenus dupliqués ou jugés trop semblables. Ceci peut survenir lorsqu'un site propose plusieurs fois le même contenu sous différentes URL, ou lorsque ce contenu est un copié-collé d'un autre site (source plus ancienne). Dans ce cas, une seule source reste indexée par Google et les autres sont considérées comme **duplicate content**.

Le *duplicate content* n'apporte pas de malus direct à votre positionnement, mais comme une seule URL est indexée, seuls les *backlinks* vers cette URL amélioreront votre positionnement, pas les autres. Il existe une solution à ce problème, la balise « canonical » qui redirige l'effet des backlinks vers une URL unique indexée par Google :

```
<link rel="canonical" href="https://www.quizzineur.be">
```

Sitemaps

Les fichiers **sitemaps** sont des fichiers (souvent au format XML) placés à la racine d'un site, reprenant toutes les URL des pages du site et d'autres informations, afin d'assister le travail d'indexation des robots.

Le pouvoir de ces fichiers est souvent surestimé par les webmasters qui ont tendance à leur attacher trop d'importance, alors que :

- Les *sitemaps* n'apportent pas de garantie d'indexation de toutes les pages qui y sont répertoriées.
- Les *sitemaps* ne remplace pas le *crawl* classique opéré par les *spiders*.
- Les *sitemaps* influent seulement l'indexation, pas le positionnement.

Le *sitemap* est donc une solution aidant l'indexation des pages par les robots, surtout utile dans la situation assez rare où votre navigation interne ne permettrait pas de trouver toutes les pages de votre site.

En conclusion, comment améliorer le référencement ?

En conclusion, le moteur de recherche est le moyen le plus employé actuellement par les visiteurs pour trouver des sites web.

Le positionnement est crucial car les premiers liens de la SERP sont de loin les plus cliqués.

Le positionnement d'une page n'est possible qu'après indexation de son URL auprès des moteurs de recherches.

Le positionnement prend en compte des centaines de critères : l'ancienneté du site, le trafic du site, la vitesse du site, l'accessibilité du site, l'adaptabilité du site, les liens entrants et les liens internes, mais surtout la présence de mots-clés.

Le choix de vos mots-clés est donc primordial et leur emplacement dans votre code HTML aussi car certains emplacements ont une importance élevée (`<title>`, `<meta name="description">`, `<h1>`, `<h2>`, `<h3>`, ``, `<a>`, etc.), mais aussi endehors du code HTML : dans vos URL (noms de domaine, de dossiers et de fichiers).

La découpe de vos sujets est tout aussi importante : il est préférable de proposer un sujet par page et une page par sujet, sans oublier d'adapter le contenu du `<head>` à chacun de ces sujets : pas question d'avoir le même *title* et la même méta-description sur toutes vos pages !

Enfin, veillez à :

1. Rendre vos pages accessibles aux moteurs de recherche ou pas (par exemple pour les pages d'administration). Possible également avec un fichier « robots.txt » configuré et placé à la racine du site.
`<meta name="robots" content="index, follow">`
2. Choisir un titre pour chaque page et rappeler le titre du site après le titre de la page :
`<title>Podium | Quizineur</title>`
3. Choisir la description du site :
`<meta name="description" content="Site de quiz informatiques sur les langages de programmation HTML, CSS, PHP, C, Python, etc.">`
4. Choisir une série de mots-clés pertinents pour chaque page (peu utile mais bon à prendre) :
`<meta name="keywords" content="quizineur, quiz, informatique, programmation, études, développement, script, C, C++, HTML, CSS, Python, JavaScript, PHP, MySQL">`
5. Géo-localiser votre entreprise/activité (si elle l'est) :
`<meta name="geo.placename" content="Mons, Hainaut">`
`<meta name="geo.region" content="BE-WHT">`
6. Travailler le contenu du site, afin de le rendre complet, ordonné et sémantique. Les éléments `<h1>`, `<h2>`, `<h3>` et `` permettent de souligner l'importance d'un contenu.
7. Optimiser votre navigation interne afin que toutes les pages soient accessibles et que les intitulés des liens contiennent vos mots-clés.

Bref, utilisez tous les moyens possibles. Pour qu'un site soit réellement visible de nos jours, il ne peut pas se contenter d'être un « site », mais doit être un super-ensemble de moyens de communication en proposant des news, de l'actualité, des échanges avec les visiteurs via des blogs ou des forums, en proposant toutes sortes de contenus (images, sons, vidéos, pdf, powerpoints), en multipliant les liens avec d'autres sites, en étant présent sur Google Maps, sur FaceBook, sur YouTube, etc., etc., etc.

Projet de site web

À rendre au plus tard le via l'eCampus

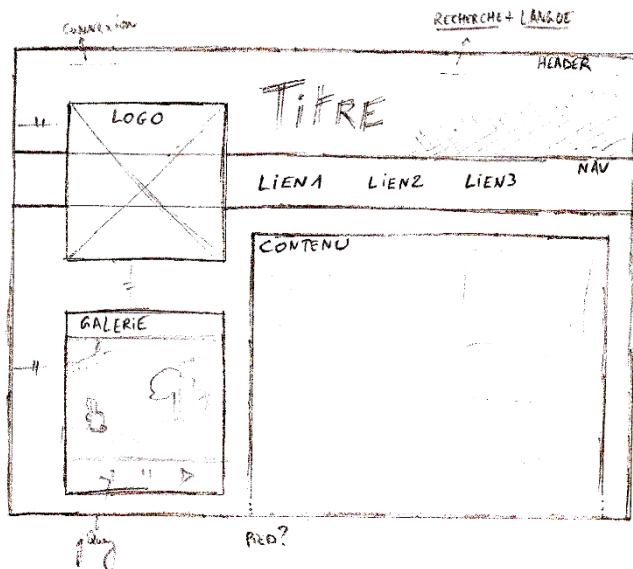
Dans le cadre de ce cours, il vous est demandé de créer un projet individuel de site web. Ce projet sera en partie développé lors des séances de travaux pratiques. Ce projet ne sera pas mis en ligne.

1. Sujet du site

Commencez par choisir le sujet de votre projet : de préférence un sujet pour lequel vous montrez de l'intérêt et que vous serez capables d'illustrer et de commenter.

Une fois le sujet choisi, identifiez le contenu de votre site : le nombre de pages et leur(s) contenu(s), la navigation, la répartition des différentes sections du sujet, la navigation, etc.

2. Maquette



Réalisez sur papier la maquette du site, c'est-à-dire un brouillon de l'apparence de votre site reprenant les différentes sections, leurs positions et leurs dimensions approximatives.

Si vous travaillez sur du papier quadrillé, cela peut vous aider à respecter l'échelle de votre maquette. Par exemple en travaillant sur une largeur de 48 carrés et en considérant que chaque carré représente 40px (=960px de large), vous obtiendrez un résultat plus proche de votre maquette.

Ensuite, réalisez une seconde maquette au format Smartphone (disons 320px), pour représenter l'agencement des informations sur un écran plus petit.

Identifiez dans cette maquette, quelles parties du site seront communes à toutes les pages et quelles parties varieront d'une page à l'autre. Il est important que la navigation et que les éléments propres à l'identité du site se retrouvent sur chaque page de manière uniforme.

Évaluation du Projet

3. Obligations (le non-respect de ces obligations entraînera la cote de 0/20 pour le projet)

- **Poids** : le site ne doit pas dépasser les 5Mo (poids du dossier compressé en .rar ou en .zip). A vous de vérifier le poids de chaque fichier et d'optimiser les plus lourds !
- **Langages** : le site doit être codé uniquement en HTML5 et en CSS. Les autres langages sont proscrits.
- **Outils** : le site doit être codé avec un éditeur de texte comme Notepad++, le bloc-notes, Smultron, Sublime Text, etc. Les éditeurs wyzwyg comme DreamWeaver, les CMS et les générateurs de code sont proscrits.
- **Travail personnel** : il est interdit d'utiliser des templates téléchargés sur le Web. Comme il s'agit ici d'un projet fictif restant dans un cadre scolaire, il vous est permis de copier des parties de textes ou d'utiliser des images trouvées sur le Web. Ceci ne concerne que les contenus, pas le code.
- **Orthographe** : le site ne doit pas contenir trop de fautes d'orthographe. Relisez scrupuleusement vos textes et si nécessaire, aidez-vous d'un correcteur orthographique.
- **Page « informations »** : prévoyez une page avec votre nom, votre email et une image de vos maquettes papier.
- **Respect des délais** : le projet doit être remis dans les temps, via Moodle et sous forme de dossier compressé .rar ou .zip (les autres formats ne sont pas acceptés).

4. Critères d'évaluation

Critères	Points	Commentaires
Fichiers : structurez votre dossier ; triez les images dans des sous-dossiers ; ne remettez que des fichiers utiles ; surveillez le poids des images ; optimisez le nom des fichiers et leur format.		
Ergonomie et Design : respectez les règles d'ergonomie vues en classe (lisibilité, proximité, affordance, etc.) et faites preuve de bon sens ; accordez votre mise en page et votre design à votre sujet et à vos besoins !		
Compatibilité : vérifiez votre site dans Edge, Chrome et Firefox (actuels).		
HTML : veillez à la lisibilité du code, à son indentation et au respect des normes du W3C (test Validator) ; choisissez les balises HTML adéquates (sémantique).		
CSS : idéalement, prévoyez un seul fichier CSS externe reprenant toute la mise en page de votre site (des CSS supplémentaires sont tolérés dans certains cas) ; <i>reset CSS</i> permis ; KNACSS permis ; veillez à l'ordre de vos déclarations et à leur répartition en sous-sections (par exemple : <i>header, nav, content, marge, footer</i>)		
Référencement : optimisez le référencement de vos pages selon les conseils vus au cours.		
Accessibilité : veillez à appliquer les principaux critères d'accessibilité		
Responsive : un comportement Responsive fonctionnel : support de toutes les définitions d'écran de largeur 320px à 1980px.		
Total		

Sources

ANDRIEU Olivier, Réussir son référencement web, éditions Eyrolles, 2018
 BEUZIT Patrick, aide-mémoire HTML 4 et CSS, éditions Eyrolles, 2002
 BOUCHER Amélie, Ergonomie Web, éditions Eyrolles, 2007
 CEDERHOLM Dan, CSS3 pour les Web Designers, éditions Eyrolles, 2012
 DRAILLARD Francis, Premiers pas en CSS et XHTML, éditions Eyrolles, 2009
 GOETTER Raphaël, CSS3 Grid Layout, éditions Eyrolles, 2019
 LAWSON Bruce & SHARP Remy, Introduction à HTML5, éditions Pearson, 2012
 MARCOTTE Ethan, Responsive Web Design, éditions Eyrolles, 2012
 MEYER A. Eric, Conception de sites Web avec les CSS, CampusPress, 2007
 NEBRA Mathieu, Réussir son site web avec XHTML et CSS, éditions Eyrolles, 2009
 PANDELE Eduard, HTML, éditions Marabout, 1997

Abondance : <https://www.abondance.com>

Alsacréations : <http://www.alsacreations.com>

Comment ça marche : <http://www.commentcamarche.net>

Développez.com : <http://www.developpez.com>

MDN : <https://developer.mozilla.org/fr/>

OpenWeb : <http://openweb.eu.org>

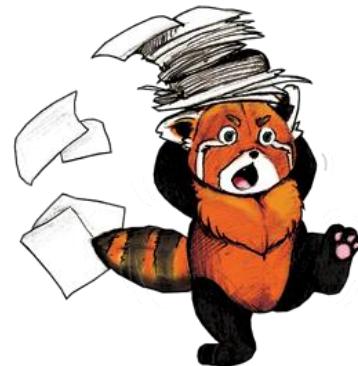
Validator W3C : <http://validator.w3.org>

W3C : <http://www.w3.org>

W3Schools : <http://www.w3schools.com>

Wikipédia : <http://fr.wikipedia.org>

Les images de la bande dessinée « Okko » sont signées Hub.



Illustrations “Firefox” par Maïlee Huynh

Ouvrages conseillés aux étudiants :

Pour une approche complète de l’HTML et du CSS :

DRAILLARD Francis, Premiers pas en HTML5 et CSS3, éditions Eyrolles, 2019

Pour approfondir vos connaissances en ergonomie d’interfaces :

BOUCHER Amélie, Ergonomie Web : Pour des sites web efficaces, 3^e édition, éditions Eyrolles, 2011

Pour tout connaître sur le référencement web :

ANDRIEU Olivier, Réussir son référencement web, éditions Eyrolles, 2018

Sites conseillés aux étudiants :

Alsacréations : <http://www.alsacreations.com>

CanIUse : <http://caniuse.com>

HTML5Test : <http://html5test.com>

MDN : <https://developer.mozilla.org/fr>

Quizineur : <https://www.heh.be/quizineur>

Validator W3C : <http://validator.w3.org>

W3Schools : <http://www.w3schools.com>