



Comment
ça marche .net

Tout sur **HTML5** et **CSS3**

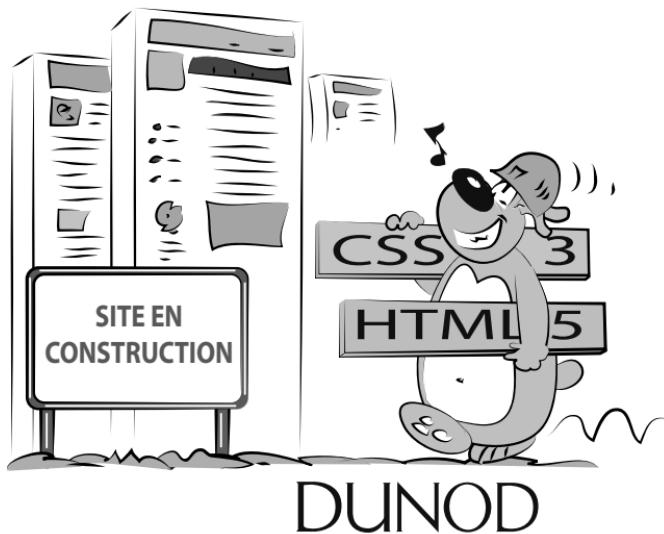


Illustration de couverture : Rachid Maraï
Maquette de couverture : MATEO
Mise en pages : ARCLEMAX

Le pictogramme qui figure ci-contre mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit, particulièrement dans le domaine de l'édition technique et universitaire, le développement massif du photocopillage. Le Code de la propriété intellectuelle du 1^{er} juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établissements



d'enseignement supérieur, provoquant une baisse brutale des achats de livres et de revues, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée. Nous rappelons donc que toute reproduction, partielle ou totale, de la présente publication est interdite sans autorisation de l'auteur, de son éditeur ou du Centre français d'exploitation du droit de copie (CFC, 20, rue des Grands-Augustins, 75006 Paris).

© Dunod, Paris, 2012
ISBN 978-2-10-058433-8

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5, 2^o et 3^o a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.



Table des matières

Avant-propos	XII
1. Historique	1
Internet	1
Web	1
Entre-temps	2
Naissance de l'HTML5	3
2. Qui fait quoi ?	5
Structures des pages	5
> HTML	5
> CSS	6
Styles	6
Animations	7
> Gif animé	7
> Flash	7
Multimédias	7
> Flash	7
> Silverlight	7
> HTML5	7

Interaction	8
> JavaScript	8
Code dynamique	8
Newsletters	9
Et pour les mobiles ?	9
3. 10 questions essentielles...	11
> Puis-je commencer dès maintenant à écrire un site en HTML5 et CSS3 ?	11
> Mon navigateur est-il compatible avec HTML5 et CSS3 ?	11
> Puis-je mélanger du code HTML4 et HTML5, CSS2 et CSS3 ?	12
> Quels sont les navigateurs compatibles avec HTML5 et CSS3 ?	12
> L'HTML5 et les CSS3 fonctionnent-ils sur tous les systèmes ?	12
> L'HTML5 améliore-t-il le référencement et le positionnement ?	12
> L'HTML5 améliore-t-il l'accessibilité des sites ?	13
> Comment reconnaître d'un coup d'œil un site fait en HTML5 ?	13
> Les sites en HTML5 peuvent-ils être lus sur mobiles ou tablettes ?	13
> Dreamweaver permet-il de faire de l'HTML5 et CSS3 ?	13
> Les CMS utilisent-ils l'HTML5 ?	13
4. Structures et balises	15
Le <!DOCTYPE>	18

Balise <html>	19
Balise <link>	19
Structuration des contenus et balisage sémantique	21
> Grandes catégories de contenus d'après le W3C	23
Les nouvelles balises	26
> Balises de Structures	26
> Exemples de Structures	31
> Les autres nouvelles balises	35
> Grand nettoyage de printemps chez HTML 4.01	42
5. Insertion de médias	45
Intégration d'une vidéo	45
> Différents codecs	45
> La balise <video>	46
> Les sources vidéos	49
Intégration d'un son	50
> Différents formats	50
> La balise <audio>	50
> Les sources audios	51
Outils gratuits	51
Intégration d'une image SVG	52
> Le langage SVG	52
> Utilisation de la balise <svg>	53
> Utilisation de la balise 	53
> Utilisation de la balise <object>	54

Intégration d'un fichier SWF	54
> Le format SWF	54
> Utilisation de la balise <embed>	55
> Utilisation de la balise <object>	56
6. Les formulaires	57
Les nouveaux champs de formulaires	57
> Les champs de saisie	58
> Les champs numériques	59
> Les champs temporels	60
Les nouvelles balises de formulaires	62
> La balise <datalist>	62
> La balise <progress>	63
> La balise <meter>	64
Les nouveaux attributs	64
> les attributs des champs numériques et temporels	64
> Les attributs d'obligation et de validation des champs de saisie	65
> Les attributs ergonomiques des champs de saisie	67
7. Crédit graphique	69
Le canevas	69
> La balise <canvas>	69
> Compatibilité des navigateurs	70
> Le type de rendu	71
> Les coordonnées du canevas	71
Dessiner des formes simples	72
> Formes rectangulaires	72

> Tracés	73
> Polygones	74
Dessiner des formes complexes	75
> Arc de cercle par utilisation d'angles	75
> Arc de cercle par utilisation de tangentes	77
> Courbes quadratiques	79
> Courbes de Bézier	80
Styliser les contours et les formes	81
> Style des contours	81
> Ajouter des couleurs	82
> Utiliser une ombre portée	86
Afficher du texte	87
Importer une image	88
> Affichage d'une image	88
> Image comme motif de remplissage	91
Transformation d'une forme	91
> Sauvegarde et restauration de l'état du canevas	92
> Rotation du canevas	93
> Déplacement du canevas	94
> Mise à l'échelle du canevas	95
> Matrice de transformation	95
Effacer des pixels	97
8. Principales API JavaScript	99
La géolocalisation	99
> Les méthodes de géolocalisation	99
> Découverte de l'interface de programmation	100

Le stockage des données	108
> Avant le HTML5	108
> Découverte de l'interface de programmation	109
> Le stockage de données par session	110
> Le stockage local de données	114
9. Positionnement & accessibilité	117
Métadonnées, Microdatas, Rich Snippets...	118
API History	119
> Que permet cette API History ?	120
L'influence de la structuration sémantique du code par les nouvelles balises HTML5	121
Type de champ « search »	122
Du nouveau dans les liens avec l'attribut <rel>	123
10. Les nouveaux sélecteurs CSS	125
Les sélecteurs CSS3	125
> Les sélecteurs d'attributs	125
> Le sélecteur d'adjacence indirecte	127
Les pseudo-classes CSS3	128
> L'énième enfant	128
> Les premiers et derniers enfants	132
> Les enfants uniques	133
> La cible d'un lien	134
> La racine du document	136

> Les éléments vides	136
> La pseudo-classe de négation	137
Les pseudo-classes CSS3 de formulaires	138
> Les champs actifs, inactifs et cochés	138
> Les champs obligatoires, valides et invalides	139
Les pseudo-éléments CSS3	140
> Le pseudo-élément de sélection	140
 I. CSS3 : enrichissements graphiques	141
Les Bordures de boîtes (<i>Border</i>)	143
> Border-radius	143
> Border-colors	144
> Border-images	144
> Box-shadow	145
Les Backgrounds	146
> Background-clip	146
> Background-Size	148
> Background Dégradés	148
> Background Multiples	149
Text-Shadow	149
Opacité	150
Transparence	151
 2. Typographie et mise en page	153
Typographie	153
> Avant et jusqu'à présent, pour l'affichage de polices, il y avait deux choix	153

> Maintenant	154
> Les formats de polices	154
> Déclaration dans la page HTML	155
> Résultat	155
Multi-colonnage	156
> Mise en œuvre	156
13. Animations avec les CSS3	159
Les transitions	159
> Les déclencheurs	160
> La propriété raccourcie transition	160
> Les propriétés uniques	161
> Les effets de l'interpolation	162
> Les interpolations multiples	164
> Les interpolations en chaîne	164
Les animations	165
> La règle CSS @keyframes	165
> La propriété raccourcie animation	167
> Les propriétés uniques	168
> Les animations multiples	168
> La propriété animation-fill-mode	169
Les transformations 2D	170
> La propriété raccourcie transform	170
> Le déplacement	170
> La rotation	171
> La mise à l'échelle	172
> L'inclinaison	172
> Une matrice de transformation	173
> La propriété transform-origin	173

> Les transitions, les animations et les transformations	174
14. CSS3 pour mobiles et tablettes	177
L'espace visualisable	178
> Les dimensions d'un support mobile	178
> Quand les pixels nous jouent des tours	178
> La balise <meta> viewport	179
Les requêtes de média	180
> Les différentes méthodes d'importation	181
> Les types de médias	182
> Les caractéristiques du support de visualisation	183
> Exemples de requêtes de média	184
Cas pratique	186
> Mise en place du HTML	186
> Les requêtes de médias	188
Index	195



Avant-propos

Avez-vous remarqué comme tout le monde parle d'HTML5 et de CSS3 ?

En revanche, les mettre en pratique est moins courant. Un peu comme avec le Web 2.0 !

On en discute, on philosophie, on se targue d'être un spécialiste, les couloirs de la cafétéria bruissent de noms de nouvelles balises, d'espoirs nouveaux portés par les CSS mais... si, au détour de quelques pages, on faisait une pause, si on prenait un tout petit peu de temps pour voir où on en est de ces fameux HTML5 et CSS3 : qui fait quoi, pour quand, avec quoi, et quelles possibilités envisager pour les prochains sites, les prochains clients ?

C'est ce que vous proposent les deux auteurs, Laurent Khouri et Jean-Marie Cochetteau, des gens du métier travaillant en indépendants, infographistes et Web développeurs. Des professionnels également de la formation Web, pratiquant une veille constante pour assurer au mieux leurs missions.

Certes, la taille de l'ouvrage est modeste, mais d'une part, il se lit plus vite et reste économique et d'autre part, il va à l'essentiel : ce qu'il est possible de mettre en œuvre, et comment se faire plaisir dès maintenant avec les nouvelles technologies, car rien n'est encore définitif dans ces domaines.

Ce livre se veut non pas être une Bible, mais un compagnon de travail, foisonnant d'exemples disponibles en téléchargements ou en test sur le site *ad hoc*, regorgeant d'astuces et d'adresses d'outils Web vous permettant d'être rapidement et facilement performant. Le tout dans un langage simple, clair et efficace.

Bonne lecture et bienvenue dans le monde fabuleux du Web 2012 !

**!**

Historique

Internet

Avant d'attaquer bille en tête l'HTML5 et les CSS3, il semble bon de faire un bref rappel des divers épisodes de l'Internet qui nous y ont conduits.

Au commencement, c'est-à-dire au siècle dernier, très précisément le 7 avril 1969, naissait Internet, précédé par son ancêtre, Arpanet, un réseau d'origine militaire, qui date de la publication du premier document numérique décrivant les caractéristiques techniques d'Internet : un RFC (*Requests For Comment*), et publié par Steve Crocke. Puis il se passa 20 ans avant que deux inventeurs de génie, Tim Berners et Robert Cailliau, ne développent une application pour Internet : le Web (mars 1989).

Web

Ce Web est représenté par un ensemble de pages, écrites avec un nouveau langage balisé, l'HTML (*Hypertext Markup Language*), pouvant contenir du texte, des images et surtout des liens, à la base de tout ceci. Important : ces pages doivent être accessibles depuis un serveur par une adresse URL.

Afin d'encadrer ce langage HTML en apportant les règles et les normes syntaxiques, sémantiques et de développement, un orga-

nisme fut créé en 1994 : le W3C pour *World Wide Web Consortium*). En 2012, c'est toujours ce même W3C qui est responsable de l'HTML5 et des CSS3.

Entre-temps

Entre-temps, le W3C a travaillé sur l'HTML à partir de 1989, pour arriver en 1995 à la publication de la norme HTML 2.0. Puis, est venu l'HTML3 en 1997 et enfin, en décembre 1997, la publication des spécificités de l'HTML 4.0 qui continue à régir une grande majorité des sites présents sur la Toile actuellement.

Déjà à l'époque, en 1997, on parlait non seulement de séparation entre le contenu et le style, mais encore de sémantique. Seulement, il faut bien se rappeler que les pages HTML sont lues par les navigateurs (*browser*) et que ceux-ci ne sont pas tenus de respecter les consignes W3C, même si la plupart des responsables de ces navigateurs siègent au sein de cet organisme. Conclusion : en fonction des lecteurs (les navigateurs), un site pourra s'afficher de diverses manières, et pas toujours de la meilleure façon qui soit d'ailleurs, surtout avec Internet Explorer.

Puis, le W3C décida de laisser en sommeil des spécificités de l'HTML4, pour s'intéresser à l'XML, censé faire évoluer le langage HTML en étant plus strict sur la syntaxe. Nous étions alors en 2001 : apparut alors le XHTML, issu de l'XML et de l'HTML qui fut suivi en 2002 par le redoutable XHTML 2.0, redoutable car plus du tout compatible avec l'XHTML1 ou l'HTML4. Et pendant ce temps-là, plus rien n'évoluait et surtout pas l'HTML4.

Du coup, en 2004, certains développeurs d'Opéra, rejoints ensuite par quelques autres de la fondation Mozilla (Firefox) et d'Apple, travaillèrent sur un projet pour tenter de faire évoluer l'HTML4. Ce groupe fut dénommé le What WG (*Web Hypertext Application Technology Working Group*).

Naissance de l'HTML5

Les travaux sur l'XHTML 2.0 n'aboutissant pas, le W3C dut définitivement abandonner ses projets en 2007, au profit des travaux initiés par le WhatWG : l'HTML5. Ce dernier fut officialisé en 2009 et en 2011 parut un document détaillant le plan d'avancement. Seul problème : l'HTML5 était encore en phase de développement et sa finalisation n'est à ce jour pas prévue avant 2014, voire 2020.

Seulement, à partir de 2010, les choses ont commencé à se préciser pour les web-développeurs, les spécifications sont devenues plus précises, des expérimentations ont pu commencer et elles ont très vite répondu à la demande et à l'engouement des développeurs, lassés des pénuries de fonctionnalités de l'HTML4 et surtout du peu d'entrain de certains navigateurs, Internet Explorer pour ne pas le nommer, à faire évoluer les choses.

Aujourd'hui, en 2012, il est désormais possible de développer en HTML5 et CSS3, à condition de pouvoir se passer d'une clientèle visionnant les sites sur des versions antérieures à Internet Explorer 9. La plupart des fonctionnalités de l'HTML5 tournent sous les navigateurs de dernière génération tels que Firefox, Chrome, Opera, Safari et Internet Explorer 9.



Qui fait quoi ?

Pour les novices du Web, designers ou développeurs, amateurs ou professionnels, mais qui sont un peu perdus avec toutes ces normes et ces langages ainsi que ces tendances et ces outils du Web sans cesse en évolution, il est peut-être bon de faire un bref récapitulatif de qui fait quoi aujourd'hui et de tenter de prévoir qui fera quoi demain. Nous ne passerons en revue que les principaux langages encore utilisés actuellement.

Structures des pages

HTML

(*Hypertext Markup Language* = langage de balisage d'hypertexte)

La notion d'hypertexte est sans doute à l'origine de tout, définissant un système où en partant d'un élément, on est « envoyé » vers un autre élément en rapport avec le 1^{er}, et ainsi de suite.

C'est le rôle du langage HTML5 désormais qui succède à l'HTML4 et au XHTML1. Il faut savoir que l'HTML est un langage interprété par les navigateurs (Internet Explorer, Firefox, Chrome...), ne pouvant en aucun cas effectuer des calculs, même simples comme par exemple une multiplication. C'est un langage simple, composé de balises qui permettent deux types d'actions essentielles :

- définir des zones de contenus (comme des boîtes avec couvercles associés, et qui peuvent s'empiler ou s'emboîter telles les « poupées russes »).

- ▶ placer des liens sur des mots, des phrases, des images, etc. permettant ainsi de naviguer dans les diverses pages d'un site, mais aussi entre les millions de sites existant sur la Toile.

L'arrivée de l'HTML5 apporte beaucoup plus de hiérarchie sémantique dans le choix de ces balises en fonction de leurs contenus.

CSS

(*Cascade Style Sheet* = feuille de style en cascade).

Même si le terme « style » apparaît dans les mots permettant de nommer ce langage, les CSS participent aussi énormément à la structure des pages, notamment dans leur partie centrale, en gérant et en définissant le multicolonnage ou en étant souvent à la base des menus. Comme l'HTML, le langage CSS est régi par le W3C, association internationale responsable de l'évolution des normes et standards de ces langages. C'est également un langage dit simple, permettant de décrire, à partir d'une syntaxe propre aux CSS, des propriétés qui seront appliquées ensuite sur les balises HTML afin de piloter les zones ou objets qu'elles définissent.

Pour écrire du code HTML ou CSS, un simple éditeur de texte suffit tel que Notepad (sous Windows) ou bien SimpleText (Mac), mais l'exercice sera facilité avec des éditeurs spécialisés tels que Notepad ++ ou Dreamweaver qui possèdent entre autres des outils de coloration syntaxiques. En tout cas, évitez absolument les logiciels type traitement de texte qui ramèneront des artefacts à votre code.

Styles

Par style, il faut entendre la mise en style des contenus :

- ▶ styles typographiques : taille, choix et couleur des polices ;
- ▶ gestion des couleurs pour les liens, les zones, les boutons ;
- ▶ gestion des fonds...

Animations

Gif animé

Au début du Web, les animations étaient succinctes en raison de leur support : les Gifs animés.

Flash

Puis vint le temps du logiciel Flash (conçu par Macromedia puis racheté par Adobe) qui donna ses lettres de noblesse aux animations sur le Web, et qui, en 2012, reste toujours leader pour les bannières publicitaires. Mais depuis quelques années, le langage JavaScript a pris la suite, surtout depuis que des bibliothèques d'animations préprogrammées sont apparues : les JQueries.

Multimédias

Par multimédias, on désigne essentiellement la gestion des objets audio et vidéo.

Flash

Depuis plusieurs années déjà, Flash s'est imposé comme le leader dans ce domaine, avec la fourniture de lecteurs (*player*) qui prennent le relais du navigateur pour lire et afficher ces médias.

Silverlight

Néanmoins, Flash est fortement concurrencé dans le domaine vidéo par un autre lecteur, produit par Microsoft : Silverlight. Cela dit, le principe reste le même, Silverlight est un lecteur qui vient prendre en charge le lecteur vidéo à la place du navigateur.

HTML5

Une des grandes nouveautés de l'HTML5 est le support en natif (donc sans avoir besoin de *player*) de certains formats dédiés à la vidéo et à l'audio. En somme, le futur du multimédia sur les sites Web.

Interaction

JavaScript

L'interactivité, qui peut exister entre l'internaute et certains éléments des pages Web, et qui rend toujours les sites plus agréables à visiter, est en majeure partie supportée par du code JavaScript. À noter qu'avec le temps et l'avancée des CSS, de nombreuses interactions jadis supportées par le JavaScript (JS) le sont désormais par les CSS, et CSS3 notamment.

Code dynamique

Pour schématiser, les sites peuvent être classés en deux catégories : ceux dont le contenu des pages est fixe et ceux dont le contenu est variable ; c'est le cas par exemple pour les sites d'e-commerce : l'affichage varie suivant le choix des visiteurs, mais également en fonction des mises à jour effectuées sur les contenus.

Rappelons que les navigateurs ne savent interpréter que de l'HTML, du CSS et du JS. Par conséquent, pour ces sites dynamiques, il leur faut un langage capable de générer ces mêmes langages. Il s'agit de PHP (*Langage Open Source*) et de .Net (qui se dit « dot Net », licence Microsoft). Ce sont tous les deux des langages évolués, « de type langages objets », et qui nécessitent un temps certain de formation. Ce sont aussi les seuls, avec l'Action Script de Flash, à pouvoir gérer les bases de données, indispensables dans le cas de formulaires par exemple et pour enregistrer des données, comme les coordonnées d'un utilisateur, la liste et les prix des produits du site.

Une remarque : peut-être avez-vous entendu parler de WordPress, Joomla, Drupal, Typo3, Spip ou encore Made Simple. Tous ces termes désignent en fait des CMS (*Content Manager System*), qui sont des applications Web capables de générer des sites. Mais ces CMS ont surtout existé grâce aux blogs, permettant à un néophyte de créer en quelques heures un espace de discussion sur le Web. Ils ont pris de l'ampleur et se sont étoffés en termes de fonctionnalités proposées, grâce, il faut le noter, à l'énorme

communauté d'utilisateurs. À tel point qu'il n'est pas rare de rencontrer des portails d'entreprise, voire des sites d'e-commerce les utiliser.

Tous ces CMS sont écrits à la base en PHP pour fournir de l'HTML, accompagné de fichiers CSS et JS. Concernant l'interface de ces sites créés par des CMS (la page qui apparaît aux visiteurs), on parle dans ce cas de thèmes (à base d'HTML et de CSS). Il en existe désormais de nombreux à base d'HTML5 pour les deux principaux CMS, à savoir WordPress et Joomla.

Newsletters

Les newsletters que vous recevez sont en fait des pages HTML, mais un peu particulières car destinées à être consultées non pas par un navigateur standard mais plutôt par des clients de messageries (puisque arrivant de la même manière qu'un mail) telles qu'Outlook, Mail, Thunderbird ou Gmail. Or, ceux-ci sont très frileux et ont une peur maladive des codes malveillants insérés dans leurs envois. Du coup, ils sont très restrictifs en termes de fonctionnalités pour ces newsletters.

Conclusion : les JS, PHP, .Net ou CSS externes sont totalement prohibés et le code HTML ressemble beaucoup plus à celui que faisaient nos grands-parents qu'à du code HTML5. Et ce n'est pas près de changer malheureusement.

Et pour les mobiles ?

Pour les mobiles, c'est-à-dire les smartphones et tablettes, les choses sont légèrement différentes, dans la mesure où les systèmes d'exploitation diffèrent quelque peu. On retrouve bien sûr les navigateurs tels que Firefox, Opera ou Chrome, et donc les pages reçues seront bien en HTML (4 ou 5), CSS et JS. Par contre, certains *players* sont malheureusement absents de modèles très répandus tels que l'iPhone et l'iPad.



10 questions essentielles...

Puis-je commencer dès maintenant à écrire un site en HTML5 et CSS3 ?

Oui, à moins que la cible de votre prochain site soit uniquement constituée d'utilisateurs accrochés aux versions Internet Explorer 6, 7 ou 8. Et même si la date annoncée de finalisation de l'HTML5 n'est pas fixée avant 2014, voir 2020 ou 2022, pas d'inquiétude, le mouvement est en marche, les navigateurs modernes dans les starting-blocks et capables d'interpréter votre code. Et si vous avez des doutes quant aux performances de la version de votre navigateur vis-à-vis d'HTML5 ou CSS, c'est ici que vous serez rassuré :

- ▶ <http://html5readiness.com/>
- ▶ <http://www.findmebyip.com/litmus>
- ▶ <http://css3test.com/>
- ▶ <http://www.caniuse.com/>
- ▶ <http://www.quirksmode.org/css/contents.html>

Et si vous souhaitez un mémento à imprimer pour vous en souvenir, c'est ici : <http://img1.websourcing.fr/files/2010/06/HTML5-infographie-tout-savoir.jpg>

Mon navigateur est-il compatible avec HTML5 et CSS3 ?

Pour le savoir, référez-vous au site : <http://fmbip.com/>

Puis-je mélanger du code HTML4 et HTML5, CSS2 et CSS3 ?

Oui, cela ne pose pas de problèmes particuliers, du moment que vous déclarez un DOCTYPE de type HTML5 :

```
<!DOCTYPE HTML>
```

Quels sont les navigateurs compatibles avec HTML5 et CSS3 ?

- Firefox 11 ;
- Opéra 11 ;
- Chrome 18 ;
- Safari 5 ;
- IE 9.

L'HTML5 et les CSS3 fonctionnent-ils sur tous les systèmes ?

Oui, aussi bien sous Windows (XP, Vista ou 7) que sous Mac IOS, voire Linux. Cela dit, ce qui compte, ce n'est pas le système d'exploitation mais le navigateur puisque c'est lui qui interprète l'HTML5 ou les CSS3.

L'HTML5 améliore-t-il le référencement et le positionnement ?

La grande nouveauté dans l'HTML5 est l'organisation de la structure du site en fonction des contenus, ce qui entraîne une meilleure architecture de ces derniers, à la fois plus logique et plus fonctionnelle. Les critères de positionnement des moteurs de recherche se portant désormais principalement sur les contenus, si ceux-ci sont mieux organisés, alors les positionnements seront gagnants. D'autres éléments, tels que les micro-formats, intégrés dans l'HTML5, sont susceptibles d'apporter également leur pierre à l'édifice.

L'HTML5 améliore-t-il l'accessibilité des sites ?

L'accessibilité est très liée aux règles de référencement naturel : donc si un site, grâce à l'HTML5, gagne en positionnement, il gagnera en accessibilité. Par ailleurs, les nouvelles balises HTML5 ont bien un rôle sémantique, ce qui facilite la lecture pour les périphériques de substitution.

Comment reconnaître d'un coup d'œil un site fait en HTML5 ?

Il suffit d'aller voir, en tout début du code source, ligne 1, la déclaration de son « Doctype ». Si vous avez ceci : <!DOCTYPE HTML>, alors vous êtes sur un site en HTML5. Et pour voir le code source d'un site, il suffit sous Firefox ou Chrome de taper Ctrl + U. Sous IE, il faut aller dans le menu « Affichage » puis « source ».

Les sites en HTML5 peuvent-ils être lus sur mobiles ou tablettes ?

Encore une fois, tout dépend du navigateur installé sur votre appareil mobile. Mais à 99,9 %, que ce soit sur système Androïd, iOS (iPhone et iPad), Windows Phone, Blackberry, il y a de fortes chances pour que cela fonctionne, les mises à jour sur mobiles étant fréquentes et les appareils récents.

Dreamweaver permet-il de faire de l'HTML5 et CSS3 ?

Qui, si vous disposez d'une version CS5 minimum de Dreamweaver. À noter que si justement vous en avez une, il existe une extension Dreamweaver, permettant sa mise à jour vis-à-vis de l'HTML5 et CSS3. L'extension se nomme Adobe Dreamweaver CS5 HTML5 Pack et est téléchargeable ici : <http://labs.adobe.com/technologies/html5pack/>

Les CMS utilisent-ils l'HTML5 ?

Oui, notamment pour les deux grands ténors des CMS : WordPress et Joomla. Nul doute que d'ici un an, tous les CMS feront de même.



Structures et balises

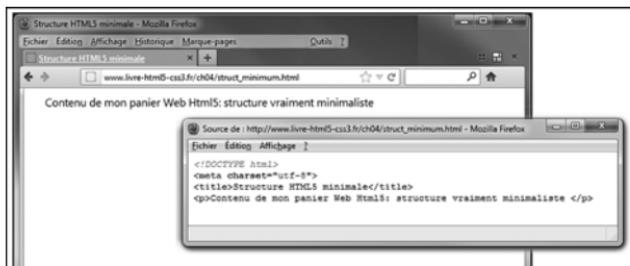
L'une des principales nouveautés apportées par l'HTML5 est sans contexte l'arrivée de balises structurelles sémantiques :

- structurelles car elles portent sur l'organisation des informations disponibles sur un site.
- sémantiques car elles permettent de cataloguer le type de ces informations.

Ces balises faisant partie du code HTML, le visiteur ne les remarque pas de prime abord, à l'inverse du navigateur et des divers robots des moteurs de recherche qui, eux, s'en rendent compte, et devraient même s'en féliciter s'ils le pouvaient, car c'est pour eux le gage d'une meilleure lecture des contenus leur permettant une meilleure interprétation.

Mais qui dit nouvelles balises ne dit pas forcément complexification du code, loin de là : la tendance serait même plutôt à la simplification.

Pour preuve, le squelette d'une page HTML5 peut désormais être résumé à quelques lignes. Ainsi, notre première page en HTML5 pourrait s'écrire de cette manière :



Et étonnamment, ce code est considéré comme valide par le validateur W3C.

The screenshot shows the W3C Markup Validation Service results page. The address bar shows "http://www.livre-html5-css3.fr/ch04/struct_minimum.html". The validation results table includes the following rows:

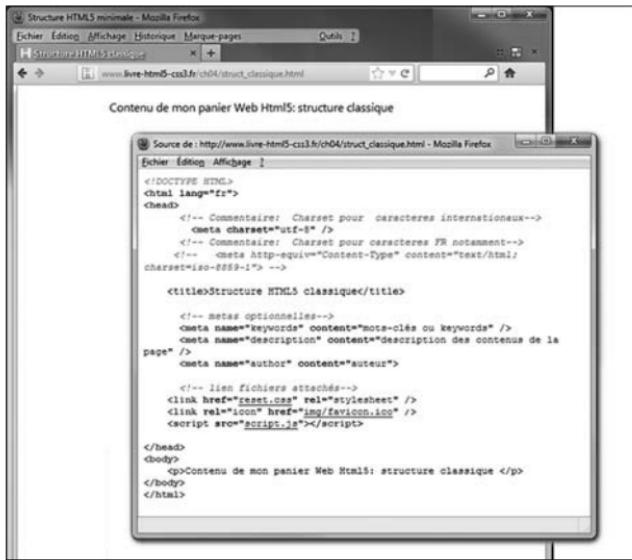
This document was successfully checked as HTML5!	
Result:	Passed.
Address:	http://www.livre-html5-css3.fr/ch04/struct_minimum.html
Modified:	(undefined)
Server:	Apache/2.2.X (OVH)
Size:	144
Content-Type:	text/html
Encoding:	UTF-8 (detect automatically)
Doctype:	HTML5 (detect automatically)
Root Element:	html

Ceux qui connaissent l'HTML 4.01 risquent bien sûr d'être un peu surpris, car ce code semble aller à l'encontre des bonnes pratiques du Web, avec l'absence de balises considérées comme primordiales dans la structure d'un code valide :

- la balise `<html>` qui encadre tout le code HTML d'une page.
- la balise `<head>` permettant de définir l'en-tête de la page, cette partie du code qui n'affiche aucun contenu mais qui, par contre, renseigne le navigateur sur tous les paramètres et les fichiers annexes (CSS, JS...) à charger en même temps que la page.
- la balise `<body>` qui se charge de renfermer tout le reste du code : contenus, menus, images, etc., ces contenus apparaissant dans le corps de la page.

En fait, même absentes du code, elles sont bien présentes dans la « pensée » des navigateurs modernes qui les supportent.

Cela dit, même si cette écriture minimalisté est validée, il est sans doute judicieux et plus prudent de garder les bonnes habitudes en continuant à inscrire ces balises pré-citées dans la structure des pages, qu'elles furent HTML 4.01 ou HTML5. Le code et vos collègues développeurs y gagneront en clarté et en compréhension car, bien sûr, elles restent toujours d'actualité et valides.



Accessoirement, on peut aussi se poser des questions concernant l'intérêt de la validité HTML par les outils W3C dont la suivante : un code validé est-il toujours meilleur ou préférable à un code non validé ? En fait, je suis d'avis de dire que la validation n'est qu'un moyen pour affirmer que votre code respecte la syntaxe et qu'il n'y a pas d'erreurs majeures, mais elle n'apporte rien au niveau de la qualité des contenus, de leur organisation, de leur hiérarchisation, de leur marquage sémantique, bref de ce qui rend un site intéressant pour le visiteur qui vient chercher une information.

The screenshot shows the W3C Markup Validation Service interface. At the top, it says "This document was successfully checked as HTML5!". Below that, there's a table of validation results:

Result:	Passed.
Address:	[http://www.livre-html5-ex1.fr/ch04/struct_classique.html]
Modified:	(undefined)
Server:	Apache/2.2.24 (Ubuntu)
Size:	460
Content-Type:	text/html
Encoding:	utf-8
Datatype:	HTML5
Root Element:	html

At the bottom, it says "The W3C validators are hosted on server technology donated by HP, and supported by community donations. [Donate](#) and help us build better tools for a better web." There is also an HP logo.

Le <!DOCTYPE>

Autre grande nouveauté de l'HTML : la déclaration DOCTYPE.

Rappelons que le DOCTYPE est le premier élément de code que l'on doit trouver dans une page HTML (avant même la balise `<html>`), qu'il est obligatoire et indispensable afin que le navigateur interprète correctement la page HTML chargée. Le DOCTYPE renseigne le navigateur sur le langage et la syntaxe employés (HTML ou XHTML) et la version de ce langage.

En version HTML 4.01, il existait 3 types de DOCTYPE suivant le mode : strict, *transitional* (transitionnel) et *frameset* (jeu de cadres), ce qui ressemblait pour le plus courant (*transitional*) à ceci en général :

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

Désormais, en HTML5, la déclaration DOCTYPE se résume à cela :

```
<!DOCTYPE html>
```

Remarquez que l'on ne précise pas la version de langage HTML.

Balise <html>

À l'instar du Doctype, la balise <html> a elle aussi subi une cure d'amaigrissement drastique passant de ceci :

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
```

À cela :

```
<html>
```

Ou si l'on y ajoute l'attribut de langue : lang="fr", on peut aussi obtenir la balise suivante :

```
<html lang="fr">.
```

À ce propos, il est important de ne pas commettre de confusion entre l'usage de cet attribut lang="fr", et celui apporté par le 'charset'.

L'attribut lang="fr" dans la balise <html> désigne la langue principale de la page. Placé ainsi dans l'en-tête, il va servir d'indicateur aux moteurs de recherche, mais aussi aux dictionnaires ou aux interfaces d'accessibilité à destination des handicapés malvoyants, tels que les synthétiseurs de voix. (L'attribut peut aussi être placé de manière plus localisé, au sein d'une balise <a href> par exemple, lorsque le lien envoie vers une page dans une autre langue).

Par contre, le charset permet de définir le type d'encodage des caractères utilisés : internationaux (utf-8) ou locaux (iso8859-1, iso8859-15 pour notre langue française par exemple). Avec une déclaration <html lang= "fr">, il est en fait possible d'avoir l'un ou l'autre de ces encodages. Si vous n'êtes pas sûr, utilisez l'utf-8 qui fonctionne pour toutes les langues. Pour plus de renseignements, notamment les avantages et inconvénients de chacun, je vous engage à consulter cet excellent article d'Alsacreations : <http://www.alsacreations.com/astuce/lire/34-charset-iso-8859-1-iso-8859-15-utf-8-lequel-choisir.html>.

Balise <link>

Parmi les autres éléments se voyant débarrassés d'attributs pesants, nous trouvons la balise <link>. Cette balise, que l'on peut

traduire par le terme « lien », permet au navigateur, lorsqu'elle est placée au sein de la balise `<head>`, d'aller charger les fichiers de code annexes à la page HTML, notamment le ou les fichiers de feuilles de style (.css) ou des fichiers de code JavaScript (.js). Désormais, en HTML5, il n'est plus nécessaire de préciser le « type » de code.

Ce qui avant s'écrivait :

```
<link href="reset.css" rel="stylesheet" type="text/css" />
<script type="text/javascript" src="script.js" ></script>
```

S'écrit désormais :

```
<link href="reset.css" rel="stylesheet" type="text/css" />
<script src="script.js"></script>
```



À savoir

Allégement du code

Certaines balises appelées « autofermantes » nécessitent un « / » avant le dernier caractère « > », telle que la balise « img » par exemple. Ce « / » est désormais optionnel. On peut donc écrire désormais :

```

```

au lieu de :

```

```

L'HTML5 autorise également l'omission de balises fermantes pour certaines suivant les conditions décrites ici, en anglais :

(<http://www.w3.org/TR/html5/syntax.html#optional-tags>)

- head
- body
- li
- p
- tr
- th

Cela dit, ce n'est pas la peine de changer vos pratiques, si vous aviez la (bonne) habitude de toujours écrire, aussitôt la balise ouvrante inscrite, la balise fermante correspondante. Pas la peine non plus de changer les paramètres de réglages de Dreamweaver ou d'autres éditeurs HTML, qui disposent de fonction d'autocomplétion, c'est-à-dire la même chose mais réalisée par le logiciel.

Structuration des contenus et balisage sémantique

Architecturer les pages Web par un balisage sémantique présente de multiples intérêts. Cela permet de :

- ▶ hiérarchiser l'information ;
- ▶ uniformiser, simplifier et affiner la structure des pages ;
- ▶ faciliter l'analyse et la lisibilité du code ;
- ▶ permettre une meilleure analyse des contenus éditoriaux par les robots de référencement ;
- ▶ ce qui, à terme, ne peut qu'améliorer le positionnement et l'accessibilité aux handicapés.

Mais qu'entend-t-on par ce terme « sémantique » ?

Wikipédia explique que c'est une branche de la linguistique qui étudie les signifiés, c'est-à-dire la représentation mentale d'une chose.



Attention !

À ne pas confondre sémantique et syntaxe, ce dernier étant un terme que l'on retrouve aussi très souvent dans le domaine informatique.

Comme le dit si justement à nouveau Wikipédia, « il y a entre la sémantique et la syntaxe, le même rapport qu'entre le fond et la forme. » La sémantique dans le domaine de l'HTML5 va donc permettre d'attacher une notion de sens commun au contenu d'une balise, tandis que la syntaxe rassemble les règles d'écriture correctes dans l'écriture du code.

En d'autres termes, ces balises à vocation sémantique permettent de préfigurer du contenu, rien qu'à l'évocation de leurs noms.

Mais attention, cette nouvelle organisation proposée par le W3C avec l'HTML5 est une vraie révolution : ce ne sont plus les spécificités ou les propriétés techniques et fonctionnelles de telle ou telle balise qui font désormais loi mais ce sont désormais leur contenu.

Le balisage des pages est à présent dépendant du sens et du rôle de chaque contenu présent dans la page. Ainsi, il va falloir que le web développeur lise et comprenne les contenus qu'il va mettre en ligne. Encore faut-il, me direz-vous, qu'il les ait..., que le client les lui ait donnés, ce qui, expérience à l'appui, est loin d'être toujours le cas, malheureusement !

Autant vous prévenir de suite, cette nouvelle organisation n'est pas si facile à appréhender au premier abord, et la première classification proposée par le W3C qui suit risque de vous faire regretter le fait de vous y intéresser. Ce serait une erreur : rappelez-vous que le fil conducteur reste le sens des contenus. Aidez-vous des divers schémas d'exemples structurels proposés dans ces pages, cantonnez-vous aux balises les plus importantes (que l'on a mis en gras dans la classification suivante) et réfléchissez au sens des contenus, car une fois que vous avez compris que ce sont eux qui priment, c'est gagné.

Notez que l'HTML4 proposait déjà quelques balises à vocation sémantique ou presque, comme les balises `<h1>`, ... `<h6>` qui préfiguraient un contenu destiné à des titres ou des sous-titres, la balise `<blockquote>` pour des citations, ou bien encore les balises `` pour des listes ordonnées.

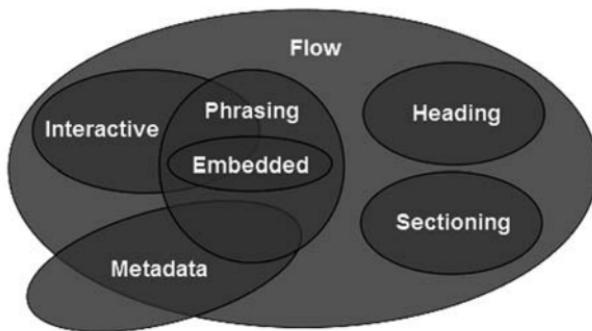
Et il faut reconnaître que les web développeurs ont conscience, depuis quelques années déjà, qu'il est important d'amener de la sémantique dans leur code, non pas forcément dans le but unique d'améliorer le positionnement (bien que...) mais surtout dans celui de lui apporter plus de clarté. Pour cela, ils ont pris l'habitude d'attacher des ID ou des Classes porteurs d'intitulés évocateurs (tels que *header*, *menu*, *navigation*, *container*, *main*, *footer*...), aux principales balises structurelles.

Du coup, le W3C, l'organisme en charge de l'HTML5 s'est appuyé sur ces habitudes de codeurs, pour déterminer leur choix en matière de création de nouvelles balises. Pour y avoir accès, ils se sont appuyés sur les données fournies par les robots d'indexation (*crawlers*) de Google mais aussi sur celui d'Opéra. (Opera a créé son propre robot, non pas dans un but de positionnement, mais plutôt pour analyser le codage des pages qui circulent sur le Web et mesurer ainsi la proportion de pages mal codées, ces éléments leur permettant de tirer des enseignements pour le développement de leur version future de navigateur).

Résultats : les termes « footer, menu, content, header, container, logo, main, copyright, sidebar, banner, navigation » viennent en tête.

Grandes catégories de contenus d'après le W3C

- Avant de voir plus en détail ces nouvelles balises, penchons-nous sur la nouvelle organisation des contenus proposée par le W3C. Désormais, tous les contenus peuvent être classés dans une ou plusieurs catégories parmi celles présentes sur ce schéma. Ces contenus sont définis par les balises qui les caractérisent.



□ Contenus type métadonnées (Metadata)

Ce sont des contenus permettant l'accès, la mise en « scène », la stylisation ou la présentation des autres contenus de la page, et qui se placent pour la plupart d'entre eux dans la partie d'en-tête de la page, délimitée par la balise `<head>` (en gras, figurent les balises les plus « populaires »). Cela concerne les balises : base, command, link, meta, noscript, script, style, title.

□ Contenus de flux généraux (flow)

En fait, ce sont tous les contenus que l'on retrouve dans le corps d'une page HTML et qui suivent les règles d'écoulement classique des flux dits « courants ». (Une balise `<div>` va, par exemple, se placer automatiquement à la suite de la balise précédente, si

aucune intervention de positionnement CSS ne se manifeste pour détourner ce flux).



À savoir

Notez à ce sujet, qu'en HTML4, on classe plutôt les balises en deux grandes catégories : les balises de type « bloc » (*block*) et les balises de type « en-ligne » (*inline*). Les balises `<div>` ou `<p>` par exemple font partie des balises de type « bloc » car elles vont se placer les unes sous les autres si on ne modifie pas le flux courant par une CSS. A *contrario*, les balises `` par exemple ou `` sont cataloguées en tant que « en-ligne », car elles ne se placent pas sous la balise précédente.

Avec l'HTML5, cette classification très « comportementale » semble laisser la place, et c'est plutôt bien, à une classification justement très sémantique, où les balises servent d'indicateurs de sens de contenus.

On y retrouve les balises suivantes (en gras, les balises les plus « populaires ») :

a, abbr, address, area, **article**, aside, audio, b, bdi, bdo, **blockquote**, br, button, canvas, cite, code, command, datalist, del, details, dfn, div, dl, em, embed, fieldset, figure, footer, form, h1, h2, h3, h4, h5, h6, header, hgroup, hr, i, iframe, img, input, ins, kbd, keygen, label, map, mark, math, menu, meter, nav, noscript, object, ol, output, p, pre, progress, q, ruby, s, samp, script, section, select, small, span, strong, style, sub, sup, svg, table, textarea, time, u, ul, var, video, wbr.

- Contenus de plans ou rubriques (au sens plan de document) (*sectioning*)

Ce sont les éléments participant à l'élaboration du schéma organisationnel de l'information de la page, permettant de signaler les diverses parties sémantiques.

Ces balises si importantes sont : **article**, **aside**, **nav**, **section**. On pourrait les qualifier de « structurantes » car elles vont être à la base de la nouvelle organisation du balisage HTML5 et de la

nouvelle structuration des contenus. On pourrait presque dire qu'elles sont les dignes successeurs des balises <div>.

Certains pourraient s'étonner de ne pas y voir figurer les balises **header** et **footer**. En effet, pourquoi pas, mais même si elles ont évidemment un rôle structurant indéniable, elles n'interviennent pas forcément au niveau de la globalité de la page comme pour les précédentes, mais peuvent également être présentes, et c'est un énorme avantage à des niveaux de structures plus fins, au sein des balises articles ou section par exemple. En ne les cantonnant pas dans cette catégorie très spécifique de « headings », il est clair que le W3C a voulu les mettre en avant, en leur donnant plus de liberté et d'importance, et en leur permettant ainsi d'être présentes à tous les niveaux.

❑ Contenus d'en-tête (*headings*)

Intitulé de contenus permettant de définir l'en-tête d'une rubrique telle que définie ci-dessus.

Balises : **h1**, **h2**, **h3**, **h4**, **h5**, **h6**, **hgroup**.

❑ Contenus de libellés (*phrasing*)

Désigne les divers contenus (pas uniquement des textes) du corps du document.

La catégorie la plus riche avec ces balises définies est la suivante :

a (*), abbr, **area**, **audio**, b, bdi, bdo, br, button, **canvas**, cite, code, command, datalist, del (*), dfn, em, **embed**, i, **iframe**, **img**, input, ins (*), kbd, keygen, label, map (*), mark, math, meter, noscript, object, output, progress, q, ruby, s, samp, script, select, small, **span**, strong, sub, sup, **svg**, textarea, time, u, var, **video**, wbr.

(*) = si la balise ne contient que du contenu de type texte.

❑ Contenu embarqué (*embedded*)

Après les contenus textuels, voici tous les autres types de contenus tels qu'audio, vidéo, tracé vectoriel... Ces contenus sont la plupart du temps encapsulés dans la page, et nécessitent un *player* mais cette fois-ci fourni directement par l'HTML5, et non plus en externe comme avant. Les balises utilisées sont : **audio**, **canvas**, **embed**, **iframe**, **img**, **math**, **object**, **svg**, **video**.

□ Contenu interactif (*interactive*)

Désigne un contenu spécifiquement destiné à une interaction avec l'internaute. Les balises nommées sont : **a**, **audio**, button, details, **embed**, iframe, **img**, **input**, keygen, **label**, menu, object, **select**, textarea, **video**

Les nouvelles balises

Balises de Structures

□ <header>

(Catégorie : *flow*)

Réduire la définition de cette balise au concept d'en-tête de page serait faux et très réducteur, tout comme croire que cela équivaudrait à ce que l'on faisait en HTML4 avec par exemple :

```
<div id="header">
    <div id="logo">.... </div>
    <ul id="menu">
        <li> menu1 </li>
        <li> ....</li>
    </ul>
</div>
```

En effet, à la différence de l'utilisation précédente, la balise `<header>` peut être présente à de multiples reprises dans la page et être la première rencontrée pour chaque élément différencié de la page (section, article, colonne...) afin d'apporter les éléments d'introduction, d'en-tête, de présentation, voire une table des matières pour chacun. Cela pourrait se traduire dans un schéma de page HTML5 comme ceci :

```
<header>
    <h1> Titre Page </h1>
</header>
<article>
    <header>.
        <h2> titre de l'Article</h2>
    </header>
    <p>.....</p>
```

```
| </article>
```

□ <footer>

[Catégorie : *flow*]

La balise *footer* est à la conclusion d'une section, d'une page ou d'un article ce qu'est la balise *header* à l'introduction ou la présentation de contenus des mêmes éléments. Comme cette dernière, elle est conçue pour être placée, de manière multiple, dans les différents nouveaux éléments structurants HTML5 de la page tels que `<article>`, `<section>`, etc., et représentera « le pied » de la balise parente ancêtre ou de la racine de la page, dans laquelle la balise `<footer>` est présente. Ainsi, au lieu d'écrire ceci :

```
| <div id="footer">
|   <ul>
|     <li>Informations légales</li>
|     <li>Contacts</li>
|     <li>Où sommes nous ?</li>
|   </ul>
|   <div>
```

On écrira désormais :

```
| <footer>
|   <ul>
|     <li>Informations légales</li>
|     <li>Contacts</li>
|     <li>Où sommes nous ?</li>
|   </ul>
| </footer>
```

Prenons un autre exemple. Plutôt que d'écrire ceci :

```
| <div id="article_Tom">
|   <p> ..... article .... </p>
|   <ul id="reference">
|     <li>Auteur</li>
|     <li>Vos réactions</li>
|   </ul>
| </div>
```

On écrira dorénavant cela :

```
| <article>
|   <p> ..... article .... </p>
|   <footer>
```

```
<ul>
    <li>Auteur</li>
    <li>Vos réactions</li>
</ul>
</footer>
</article>
```

□ <nav> :

(Catégorie : *flow & sectioning*)

« Nav » pour « navigation » vous l'aviez deviné !

Cette balise inclut donc des éléments de navigation, plutôt de nature « navigation principale » de la page, et plutôt réservés à des liens de type interne.

Exemple :

```
<nav>
<ul>
    <li><a href="index.htm">Accueil</a></li>
    <li><a href="produits.htm">Produits</a></li>
    <li><a href="contacts.htm">Contacts</a></li>
</ul>
</nav>
```

□ <article>

(Catégorie : *flow*)

Avec cette balise, les choses se compliquent, car c'est un faux-amis et il ne faudrait surtout pas croire que c'est prévu pour recevoir un article, au sens article de journaux. En fait, sa véritable vocation est de recevoir un contenu texte « remarquable » qui, bien qu'en rapport avec le reste des contenus éditoriaux de la page, peut être autonome et se suffire à lui-même. Ce peut être un article de blog, un commentaire, un message, une annonce. Mais ce n'est pas non plus un résumé, ni un chapeau d'article ; l'élément, même remarquable, reste discret.

Bref, la balise `<article>` est idéale pour encadrer et structurer tout ce qui peut se lire seul, sans l'environnement éditorial de la page. Elle permet à la fois de structurer les contenus mais également d'optimiser les fonctionnalités de recherche tout en facilitant la navigation.

```
<section id="main">
  <article>
    <H1>
      <a href="http://www.monsite.com"> La nouvelle version
      Firefox aime l'HTML5</a>
    </H1>
    <p>
      Lorem ipsum dolor sit amet, consectetur adipiscing elit.
      Vivamus elit ante, ullamcorper nec posuere.
    </p>
  </article>
</section>
```

□ <section>

(Catégorie : *flow & sectioning*)

Encore un faux-amis, car son intitulé laisse penser qu'elle est adaptée pour élaborer la structure d'une page, en définissant ses grandes parties, comme on le fait avec la balise `<div>`. Eh bien non, car le W3C indique que la balise ne doit pas être utilisée à des fins génériques, pour une mise en page par exemple, ou pour faciliter un script. Dans ce cas, c'est bien la balise `<div>` qui reste recommandée.

La balise `<section>` est plutôt réservée au regroupement d'éléments traitant de la même thématique et délimitant de grandes zones génériques, comme la mise en place de chapitres, de dossiers ou d'articles conséquents. La première balise enfant rencontrée sera, dans la plupart des cas, une balise de type `<h1 ...>`. Son emploi reste donc encore assez flou et mal défini, surtout qu'il n'est pas rare de voir du code avec la balise `<article>` englobant la balise `<section>`, et inversement. L'analyse du contenu est donc particulièrement importante dans ce cas.

Voici un exemple qui est fourni sur le site du W3C et qui illustre bien l'un des emplois : il s'agit d'un extrait de livre, où les chapitres et annexes sont définis par des balises `<section>`, l'ensemble étant regroupé dans une balise `<article>`, tout en sachant que le document final comporte plusieurs `<article>`.

```
<article class="book">
  <header>
    <h1>My Book</h1>
    <h2>A sample with not much content</h2>
```

```
<p>Published by Dummy Publicorp Ltd.</p>
</header>
<section class="chapter">
    <h1>Mon 1er Chapitre</h1>
    <p>Je ne dis pas tout dans mon premier chapitre</p>
    <p>Même avec deux paragraphes</p>
</section>
<section class="chapter">
    <h1>Continuons avec le deuxième chapitre</h1>
    <p>Bla bla bla. Vlan ! Damned, que d'actions !</p>
</section>
<section class="appendix">
    <h1>Annexe A</h1>
    <p>Le coin de mes bonnes adresses</p>
</section>
</article>
```

□ <aside>

(Catégorie : *sectioning*)

La balise `<aside>` définit un contenu qui peut être considéré comme une annexe au contenu principal (contenu tangentiel suivant les termes du W3C, au contenu principal), et qui peut être autonome, comme pour `<article>`. Si jusque-là c'est encore assez clair, cela se complique suivant que cette balise sera une balise enfant d'une balise `<article>` ou non.

Dans le 1^{er} cas, le contenu de la balise `<aside>` doit être en rapport avec celui de sa balise parente `<article>`. C'est le cas, par exemple, pour des contenus d'éléments annexes tels que des liens, des commentaires, des références bibliographiques pour un article par exemple.

```
<article>
    <section>
        <h1>Titre article</h1>
        <p>Jellentesque habitant morbi tristique senectus ....</p>
    </section>
    <section class="appendix">
        <h1>Annexe A</h1>
        <p>Le coin de mes bonnes adresses</p>
    </section>
    <aside>
        <h1>Bibliographie</h1>
```

```
<ul>
<li>Maecenas pulvinar ...
<li>elit eget auctor posuere...
</ul>
</aside>
</article>
```

Par contre, si la balise `<aside>` est externe à toute balise `<article>`, elle doit alors être considérée comme une balise permettant d'inclure des éléments annexes à la page en général mais toutefois en relation avec ceux-ci. Dans ce cas, son utilisation est similaire à ce qu'on le fait dans cet ouvrage papier, avec les encadrés (encarts textes) du type : « Astuces », « À Savoir ».

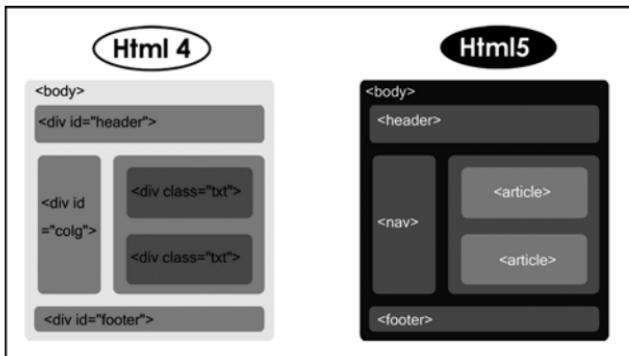
Autre utilisation : une navigation secondaire (barre latérale), encart pub, nuage de tags...

```
<aside>
<h1>Dates à retenir</h1>
<ul>
<li><a href="/calandar/2012/jan.htm">Janv 2012</a></li>
<li><a href="/calandar/2012/fev.htm">Fev 2012</a></li>
<li><a href="/calandar/2012/mar.htm">Mar 2012</a></li>
</ul>
</aside>
<article>
....
<article>
```

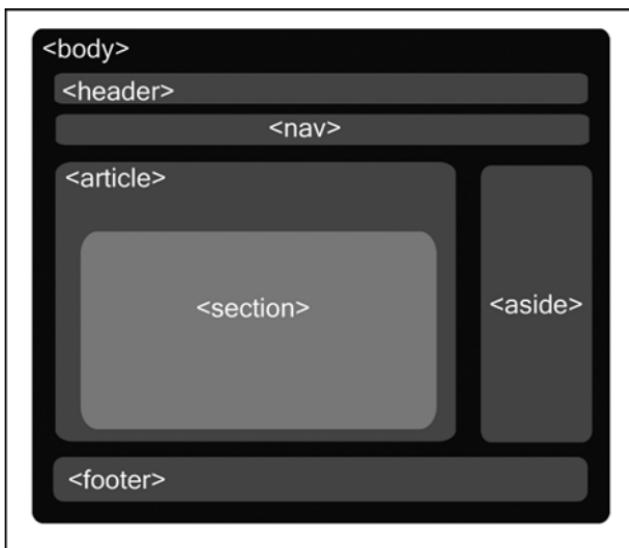
Exemples de Structures

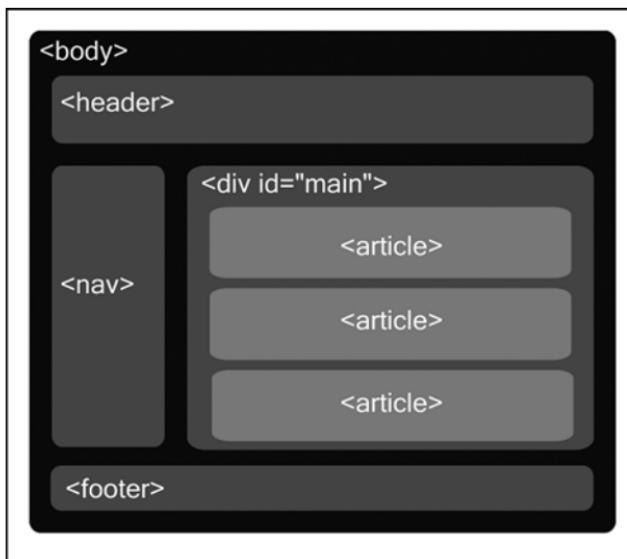
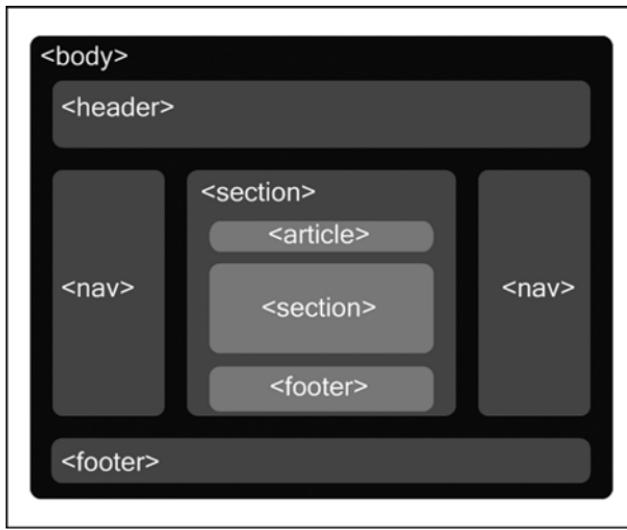
En HTML5, on remarque que l'emploi de telle ou telle balise est plus que jamais fonction de son environnement, de l'interprétation sémantique des contenus déployés et de l'expérience du développeur. Pour vous aider à y voir plus clair, voici quelques exemples de structures relevées sur des sites actuels en HTML5 qui peuvent toutes être considérées comme valides.

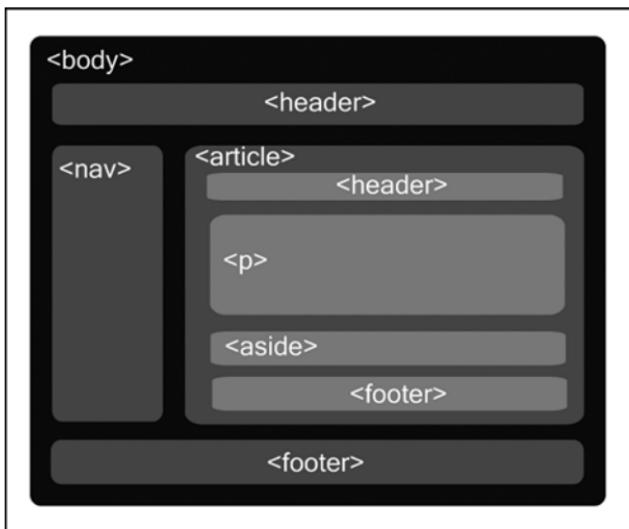
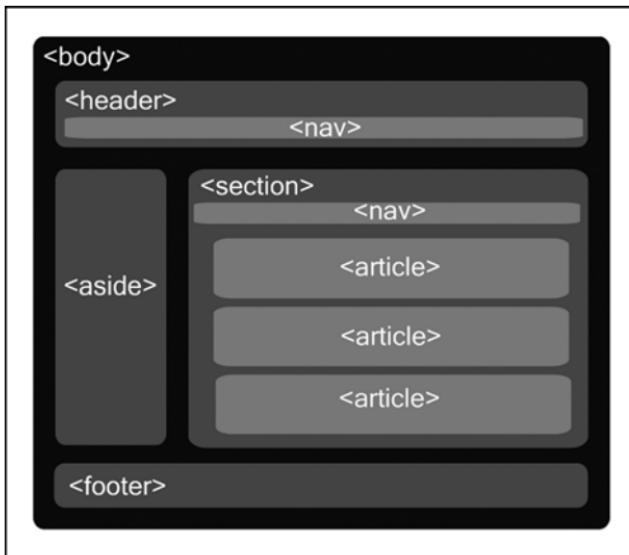
Premier exemple avec la comparaison de la même structure écrite pour l'HTML4 et l'HTML5.

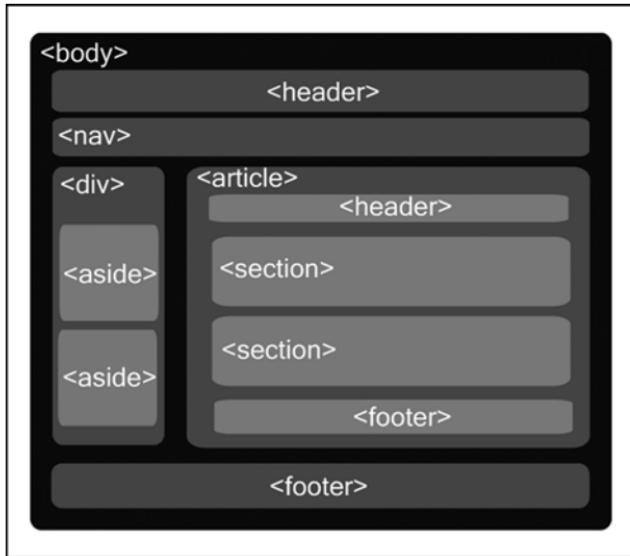


Autres exemples :









Les autres nouvelles balises

❑ <hgroup>

(Catégorie : *flow/headings*)

Les avis sont partagés sur l'utilisation de cette balise, car son cadre reste assez flou et il en faut peu pour mal interpréter son utilisation. Certains l'associent à la représentation des sommaires car elle permet d'intégrer toutes les balises de type <h1> à <h6>. En effet, cela est approprié mais il ne faut pas oublier que ces balises ne permettent pas de créer et d'insérer une table des matières mais aident plutôt à la création d'un sommaire. C'est d'ailleurs ce qu'indique le W3C dans ces documents : <http://dev.w3.org/html5/spec/the-hgroup-element.html#the-hgroup-element>)

La balise <hgroup> ne sert donc qu'aux outils de recherche ou de création, s'intéressant aux contenus titres et sous-titres. Elle permet de leur indiquer quel élément de titre doit être pris en compte dans une suite de titres et sous-titres. Pourquoi a-t-on

besoin de cette balise ? On peut se poser la question si on se place dans la logique HTML4, où il existe (ou devrait exister) un ordre hiérarchique naturellement fourni par la suite des balises `<h1>` à `<h6>`. Mais en HTML5, tout change, car avec les balises `<section>` et `<article>`, on peut désormais inclure plusieurs balises `<h..>` sans aucun risque de confusion, chaque section ayant sa hiérarchie. En revanche, il faut donc désormais un moyen pour hiérarchiser ses titres et sous-titres pour les outils de recherche.

Pour résumer : si vous pensez que vous n'avez pas besoin de cette balise, que son emploi reste accessoire ou que cela reste du délire de l'intellectuel du Web, vous risquez de déchanter dans l'avenir. Il y a fort à parier que son utilisation va devenir incontournable.

Voici un premier exemple simple :

```
<hgroup>
  <h1>Les tendances 2011</h1>
  <h2>Place aux typos</h2>
</hgroup>
<hgroup>
  <h1>Les tendances 2012</h1>
  <h2>LA sémantique arrive</h2>
</hgroup>
```

Au final, un logiciel de création de table des matières affichera :

- les tendances 2011 ;
- les tendances 2012.

Autre exemple plus complexe :

```
<hgroup>
  <h1> Les tendances Web</h1>
    <h2>Exposé en 4 tableaux </h2>
</hgroup>
  <h2> Les tendances 2011</h2>
    <hgroup>
      <h3>Site e-commerce </h3>
        <h4>Réalisés ou non avec des CMS</h4>
    </hgroup>
  <h2> Les tendances 2012</h2>
    <hgroup>
      <h4>Site e-commerce:</h4>
      <h3>Annexes</h3>
    </hgroup>
```

Cette fois, il en ressortira ceci dans un outil de création de sommaire.

- Tendances Web
 - Les tendances 2011
Site e-commerce
 - Les tendances 2012
Site e-commerce

Par contre, la page Web affichera bien cela :

Les tendances Web

Exposé en 4 tableaux

Les tendances 2011

Site e-commerce

Réalisés ou non avec des CMS

Les tendances 2012

Site e-commerce:

Annexes

- <figure> & <figcaption>

(Catégorie : *flow*)

Même si elle est plutôt destinée aux images, la balise « figure » est réservée au « contenu autonome » (*sic W3C*) tels que des images, des diagrammes, des vidéos, etc. Elle est souvent associée à la

balise `<figcaption>` qui se charge d'insérer la légende du contenu de la balise `<figure>`, le tout pouvant bien sûr être ensuite mis en forme par une feuille de style, comme c'est le cas dans la figure ci-dessous.

```
<figure>
    <!-- Placement automatique d'une image par le site
    'lorempixel.com'-->
    
    <figcaption>
        <h3>Titre de l'image </h3>
        <p>- La légende ici -</p>
    </figcaption>
</figure>
```



Titre de l'image

- La légende ici -

□ `<audio>` & `<video>`

(Catégorie : *phrasing & interactive*)

Très attendues car elles sont synonymes de libération de ces types de médias des players propriétaires genre Flash Player, Realplayer ou autre joyeuseté : QuickTime.

De plus, une grande liberté est laissée aux intégrateurs puisque ces deux éléments fournissent une API permettant de mettre en place sa propre interface de lecture ou plus simple encore,

d'utiliser l'interface fournie par le navigateur. Tous les détails figurent au chapitre 5.

□ <canvas>

(Catégorie : *phrasing*)

Sans doute l'une des balises qui sera le plus mise à l'honneur dans les années à venir, puisqu'elle permet d'afficher de manière dynamique des graphiques de type bitmap (en opposition avec les graphiques vectoriels) au sein des pages Web. C'est ce qu'attendaient tous les développeurs pour intégrer des schémas, des graphiques de type charts, mais aussi et surtout je dirais, par rapport aux plateformes mobiles des interfaces de jeux ! Tous les détails sont explicités au chapitre 7.

□ <dialog>

Pas de faux-amis cette fois-ci, l'idée est bien de signaler des conversations comme par exemple :

```
<dialog>
  <dt> Bonjour
  <dd> Ravi de vous rencontrer.
  <dt> Comment allez-vous ?
</dialog>
```

(Les balises *<dt>* et *<dd>* sont des balises de définition, rarement employées mais bien antérieures à l'HTML5).

□ <address>

(Catégorie : *flow*)

À inclure de préférence dans une balise *<footer>* pour y mettre à disposition des informations de contacts (auteurs, directeur de publication, webmaster...)

□ <mark>

(Catégorie : *phrasing*)

Une nouvelle balise qui vient compléter celles déjà existantes permettant d'attirer l'attention du lecteur sur un mot, un terme. Jusqu'à présent, nous disposions des balises **, ** et **. Et déjà, nombreuses sont les confusions d'emplois de ces diverses balises qui répondent *a priori* au même but : mettre

un mot, un terme, un vocable en valeur. Du coup, nous allons nous y attarder un peu, afin de bien clarifier les choses.

La balise `<mark>` se différencie de ses précédentes comparses, par le fait qu'elle ne sert pas à marquer l'importance d'un vocabulaire (en tant que sens apporté par ce dernier), mais s'utilise pour faire juste ressortir la pertinence d'un terme qui a été choisi (non pas pour son sens qui importe peu, mais parce qu'il est le résultat d'une sélection). Il a été sélectionné, donc il doit être mis en évidence. C'est le cas, par exemple, lorsque l'on souhaite afficher les résultats d'une recherche, ou faire ressortir une date sur un calendrier. Par ailleurs, un effet de surlignage jaune sera automatiquement appliqué par le navigateur (du moins pour les navigateurs de dernière génération) sans ajout d'une CSS.

Résultats de votre recherche sur le terme : nouveauté

Les diverses balises en "HTML"

- nouveauté la balise "mark" : [en savoir plus](#)
- nouveauté la balise "header" : [en savoir plus](#)
- Les balises "Strong" ne sont pas une nouveauté : [en savoir plus](#)

Notez bien qu'en HTML4, on aurait pu employer la balise `` avec une CSS associée, une balise ``, une balise `` ou même une balise ``. Pourquoi, dans ce cas présent où l'on veut faciliter la lecture en faisant ressortir un élément, utiliser cette balise `` ?

- la balise `` n'a aucune valeur sémantique et va donc à l'encontre de la tendance HTML5 lorsque l'on veut mettre en avant un élément. De plus, il faudrait lui adjoindre une règle CSS ce qui alourdirait et brouillerait le code inutilement.
- la balise `` ne possède pas non plus de valeurs sémantiques, mais en plus, suivant les navigateurs, celle-ci peut se voir adjoindre une stylisation différente.
- la balise ``. Si nous l'utilisons, nous faisons une légère erreur, car la balise `` possède justement un sens sémantique trop fort, du genre : « ce terme inclus dans la balise `` a une valeur plus importante que les autres ter-

mes et doit être mis en exergue, car il a une influence, par son sens, sur le texte ». Il est donc à employer quand l'auteur a voulu mettre un terme en avant, pour orienter la compréhension de son texte. Notez également que les navigateurs ont pour habitude de mettre le contenu des balises `` en gras, alors qu'aucune règle ne l'impose.

- quant à la balise ``, elle est réservée au vocabulaire empreint d'une certaine emphase. Par ailleurs, son contenu est souvent mis en italique, ce qui, encore une fois, n'est consigné dans aucune règle.

Conclusion : il suffit que les navigateurs changent d'avis pour que le rendu des balises `` ou `` change d'aspect, à moins bien sûr, que vous n'ayez pris les précautions d'y adjoindre une CSS pour bloquer l'aspect de la balise.

□ `<command>`

(Catégorie : *flow & phrasing*)

Affiche un bouton de commande (*button*, *radiobutton* ou *checkbox*) que l'utilisateur peut invoquer, mais uniquement au sein de la balise `<menu>`, qui, rappelons-le, permet de créer un menu contextuel ou une ligne de barre de commande.

□ `<time>`

(Catégorie : *phrasing*)

Pour insérer ou définir une date ou un horaire.

□ `<wbr>`

(Catégorie : *phrasing*)

Force un retour-chariot (retour à la ligne), dans le cas d'un mot très long.



Attention !

C'est une balise non fermante.

□ <meter>

(Catégorie : phrasing)

Idéal lorsque l'on souhaite indiquer une mesure dans une plage de valeurs connues. Ce code, par exemple, permet d'afficher ce type de barre de progression ; malheureusement, cela n'est possible que sur Chrome (v 17.0) et Opera (v 11.62). Dommage !

```
<div>
    <meter min="0" max="100" value="20">20/100</meter>
    <meter min="0" max="100" value="80">80/100</meter>
    <meter min="0" max="100" value="50">50/100</meter>
</div>
```



□ <progress>

(Catégorie : *phrasing*)

Permet de représenter l'état d'avancement d'un événement en cours de réalisation, notamment pour illustrer un téléchargement. Vous trouverez d'excellents exemples sur ce site : <http://www.scriptol.fr/html5/progress.php>

Grand nettoyage de printemps chez HTML 4.01

Cette arrivée de l'HTML5 a été l'occasion pour le W3C, instance décisionnelle internationale sur les langages HTML et CSS, de déclarer certaines balises de l'HTML 4.01 obsolètes :

- soit parce qu'elles n'offraient qu'un service limité, et souvent du genre rendu visuel, qui peut désormais être assuré, en mieux par les CSS.

- soit parce que les navigateurs ont de leur côté évolué (et de façon conséquente pour certains d'entre eux tels que IE qui, il faut le reconnaître, avait accumulé défauts et retards depuis sa création).
- parce que les mentalités des concepteurs ont évolué également, tout comme l'enseignement, au niveau de l'éthique et des méthodologies à employer.
- parce que le référencement naturel et l'importance d'obtenir un bon positionnement dans les pages de résultats des moteurs de recherche ont pris de plus en plus d'importance dans la construction d'un site ; de ce côté-là, l'HTML5 et ses nouvelles balises ont incontestablement une portée.
- autre élément qui a depuis pris beaucoup de valeur : l'accessibilité des sites aux handicapés, laquelle n'est pas favorisée par ces anciennes balises qui auraient plutôt tendance à compliquer les situations ou à être un obstacle aux périphériques de lecture.

❑ Balises HTML 4.01 dépréciées et obsolètes

Que signifie ce terme de dépréciation ? En fait, pour des raisons de compatibilité avec les anciens sites, le W3C ne peut se permettre de supprimer purement et simplement du « catalogue » des balises, même obsolètes. Par contre, il a décidé de :

- ne plus les lister, ni les faire apparaître dans leurs documentations.
- ne plus assurer de support à leur sujet.
- ne plus les valider lors des tests par les validateurs en ligne, et même les déclarer invalides.

De fait, je me contenterai comme eux de vous lister ces balises :

```
<applet>
<basefont>
<big>
<center>
<dir>
<font>
<strike>
<tt>
<u>
```

```
| <acronym>  
| <noscript>
```

Par contre, certaines mériteraient vraiment d'être supprimées, au vu des méfaits qu'elles peuvent engendrer, notamment en termes de référencement [les robots de moteurs de recherche étant incapables alors de poursuivre leur examen du site dans lequel elles sont présentes] :

```
| Frameset  
| Frame  
| Noframes.
```

Vous manqueront-elles ?



Insertion de médias

Tout au long de ce chapitre, nous allons découvrir un HTML réellement multimédia. Depuis sa version 2, qui date de 1994, le HTML permet d'insérer des images. Mais qu'en est-il des autres médias tels que **la vidéo ou le son** ? Le HTML5 permet désormais d'intégrer des éléments multimédias sans passer par un lecteur externe au navigateur tel que Flash ou QuickTime. Enfin, la dernière évolution dans l'insertion de médias est la possibilité d'intégrer des **images vectorielles** dans une page HTML.

Intégration d'une vidéo

Différents codecs

Commençons par les choses qui fâchent ! Pour lire une vidéo, il est nécessaire de disposer d'un codec, c'est-à-dire d'un **codeur-décodeur** dans son navigateur. Le W3C dans sa spécification de la balise `<video>` parle de « codecs » mais sans les nommer précisément. **Trois codecs** ont donc été implémentés dans les principaux navigateurs :

- **H.264** : codec propriétaire soumis au paiement d'une licence après du consortium MPEG LA.
- **VP8** : codec racheté par Google, qui l'a rendu libre de droits dans le cadre du projet WebM.
- **Ogg Theora** : codec libre de droits appartenant à la fondation Xiph.org.

Les principaux navigateurs ont donc tous choisi un codec selon leurs intérêts ou leurs convictions. Ainsi, trois groupes se sont créés parmi les principaux éditeurs de navigateurs :

- **l'universaliste** : Chrome lit tous les codecs sans distinction.
- **les défenseurs des formats brevetés** : Apple avec Safari et Microsoft avec Internet Explorer ne lisent que le format H.264 au travers des conteneurs vidéo MP4 ou M4V.
- **les défenseurs du monde libre** : Firefox et Opera ne lisent que les formats libres, donc WebM et Ogg Theora.

Cette situation aboutit à créer au **minimum deux formats** pour que votre vidéo soit lue dans le navigateur de votre internaute. Au niveau de la qualité et du poids des fichiers, le H.264 est incontestablement celui qui assure les meilleures performances, mais en contrepartie, il est aussi le plus lourd. Le codec Ogg Theora offre un bon rapport qualité/poids face au H.264. Enfin, le WebM est certainement le codec du futur, alliant la qualité du H.264 et un format ouvert. Du point de vue qualitatif, il est proche du H.264 en offrant un poids souvent divisé par deux ou trois.

La balise <video>

HTML

```
<video id="videoHTML" src="video.ogg">
<p>Votre navigateur ne peut lire cette vidéo.</p>
</video>
```

Oubliez l'ancienne intégration d'une vidéo au travers du lecteur Flash ou QuickTime, ainsi que les obscures balises `<object>` ou `<embed>` souvent imbriquées et utilisez désormais la **balise <video>**. Le premier attribut auquel l'on pense est de spécifier le chemin vers le fichier vidéo, c'est le rôle de l'attribut **src**. Il est en réalité difficilement utilisable en tant qu'attribut de la balise `<video>` en raison de la multiplicité des codecs. Le paragraphe inclus dans la balise `<video>` sert à indiquer à l'internaute qu'il n'utilise pas un navigateur récent ou capable de lire les vidéos en HTML5.

□ Dimensions de la vidéo

HTML

```
<video id="videoHTML" src="video.ogg"
```

```
width="400" height="225">
<p>Votre navigateur ne peut lire cette vidéo.</p>
</video>
```

Si vous ne spécifiez pas les dimensions de la vidéo, celle-ci prend sa taille originale. Vous pouvez néanmoins définir des dimensions différentes avec les attributs **width** pour la largeur, et **height** pour la hauteur, en pixels ou en pourcentages.

□ Affichage de la vidéo

HTML

```
<video id="videoHTML" src="video.ogg"
width="400" height="225"
poster="image.jpg">
<p>Votre navigateur ne peut lire cette vidéo.</p>
</video>
```

L'attribut **poster** permet d'afficher **une image de présentation** de la vidéo avant qu'elle ne soit lue. Typiquement, cette image représente une image prise de la vidéo avec un titre. Si vous n'utilisez pas d'affiche pour votre vidéo, c'est la première image de la vidéo qui sera visible.

Il est à noter que le navigateur Internet Explorer gère mal cet attribut.

□ Lecture automatique

HTML

```
<video id="videoHTML" src="video.ogg"
width="400" height="225"
poster="image.jpg"
autoplay>
<p>Votre navigateur ne peut lire cette vidéo.</p>
</video>
```

L'attribut **autoplay** permet le démarrage automatique de la vidéo dès son chargement.

□ Lecture en boucle

HTML

```
<video id="videoHTML" src="video.ogg"
width="400" height="225"
```

```
    poster="image.jpg"
    autoplay loop>
<p>Votre navigateur ne peut lire cette vidéo.</p>
</video>
```

En intégrant l'attribut `loop` dans la balise `<video>`, la lecture de la vidéo tourne en boucle.

□ Boutons de contrôle

HTML

```
<video id="videoHTML" src="video.ogg"
width="400" height="225"
poster="image.jpg"
autoplay loop controls>
<p>Votre navigateur ne peut lire cette vidéo.</p>
</video>
```

La balise `<video>` possède un attribut **controls** permettant d'afficher des boutons de lecture et de volume à la vidéo qui **diffèrent visuellement selon les navigateurs**. Il est à noter qu'il est également possible, en recourant au JavaScript, de créer sa propre interface de lecture.

□ Préchargement de la vidéo

HTML

```
<video id="videoHTML" src="video.ogg"
width="400" height="225"
poster="image.jpg"
autoplay loop controls
preload="auto">
<p>Votre navigateur ne peut lire cette vidéo.</p>
</video>
```

Si vous avez utilisé l'attribut **autoplay**, l'attribut **preload** va préciser le type de préchargement que vous souhaitez utiliser avec votre vidéo. Les valeurs possibles de l'attribut sont :

- **valeur none** : aucun préchargement de la vidéo ne sera effectué.
- **valeur metadata** : récupération des métadonnées de la vidéo.
- **valeur auto** : téléchargement progressif de la vidéo.

La **valeur none** permet d'économiser de la bande passante, et est recommandée lorsque vous avez beaucoup de vidéos dans une même page HTML, ou encore si vous êtes sur un site dédié aux mobiles. Les métadonnées sont des **informations associées au fichier**, comme la hauteur et la largeur de la vidéo, ou encore la durée de cette dernière.

La **valeur metadata** permet de récupérer ces informations sans pour autant lancer le téléchargement progressif de la vidéo.

La **valeur auto** permet de précharger la vidéo dès le lancement de la page HTML ; à réserver donc lorsque vous disposez d'une bande passante suffisante.

Les sources vidéos

HTML

```
<video  
id="videoHTML" width="400" height="225"  
poster="image.jpg"  
autoplay loop controls  
preload="auto">  
<source src="video.mp4" type="video/mp4" />  
<source src="video.og" type="video/ogg" />  
<source src="video.webm" type="video/webm" />  
<p>Votre navigateur ne peut lire cette vidéo.</p>  
</video>
```

Les **diverses sources vidéos** vont être intégrées dans la balise `<video>` au travers de balises `<source>`. Chaque **balise <source>** cible une vidéo, et donc un codec unique. Les attributs de `<source>` sont :

- **attribut src** : chemin vers le fichier de la vidéo.
- **attribut type** : type MIME du fichier de la vidéo.

L'**ordre des balises <source> a son importance**, car le navigateur va parcourir chaque source vidéo et afficher celle qu'il lui est possible de lire. Dans notre exemple, le navigateur Firefox, qui lit les codecs WebM et Ogg Theora, affichera la vidéo dans ce dernier codec car il est placé dans la balise `<video>` avant le codec WebM.

Intégration d'un son

Différents formats

L'intégration d'un son, comme de la vidéo, nécessite de disposer de **plusieurs formats audio** pour être lus dans les principaux navigateurs. Deux principaux formats audio y sont implémentés :

- ▶ **MP3** : format breveté et soumis au paiement d'une licence.
- ▶ **Ogg Vorbis** : codec libre de droits appartenant à la fondation Xiph.org.

Il y a encore une fois trois groupes :

- ▶ **l'universaliste** : Chrome lit tous les formats audios.
- ▶ **les défenseurs des formats brevetés** : Safari et Internet Explorer acceptent uniquement le format MP3.
- ▶ **les défenseurs du monde libre** : Firefox et Opera lisent le format Ogg Vorbis.

La balise `<audio>`

HTML

```
<audio id="audioHTML" src="audio.ogg">
<p>Votre navigateur ne peut lire ce son.</p>
</audio>
```

Il est possible d'intégrer un attribut **src** dans la balise `<audio>`, mais cela ne permet de spécifier qu'un seul fichier audio, et donc d'être illisible dans certains navigateurs. Le paragraphe imbriqué dans la balise `<audio>` sert de contenu alternatif dans le cas où la version du navigateur serait obsolète. Chaque navigateur possède une interface de lecture différente.

□ La configuration du lecteur audio

HTML

```
<audio id="audioHTML" src="audio.ogg"
autoplay loop
controls preload>
<p>Votre navigateur ne peut lire ce son.</p>
</audio>
```

Tous les attributs de `<audio>` sont **identiques à la balise `<video>`**. Nous n'allons donc pas revenir sur leur fonction ou leurs valeurs possibles, référez-vous au paragraphe précédent. Ces attributs identiques sont :

- **autoplay** : lecture automatique.
- **loop** : lecture en boucle.
- **controls** : affichage des boutons de contrôle.
- **preload** : préchargement du son.

Les sources audios

HTML

```
<audio id="audioHTML"
autoplay loop
controls preload>
<source src="audio.mp3" type="audio/mpeg" />
<source src="audio.ogg" type="audio/ogg" />
<p>Votre navigateur ne peut lire ce son.</p>
</audio>
```

Pour être lue dans les principaux navigateurs, la balise `<audio>` doit disposer de **plusieurs formats audios**. Le navigateur va utiliser le format audio qu'il peut lire.

Outils gratuits

Dans la jungle des codecs, attardons-nous sur les outils gratuits permettant de créer des fichiers avec les codecs attendus par l'HTML5.

Miro Video Converter

Disponible à l'adresse www.mirovideoconverter.com, ce logiciel est simple et efficace pour créer des vidéos compatibles avec l'HTML5.

Video Lan Client

Le célèbre lecteur audio et vidéo sait aussi convertir les vidéos et les sons dans les formats HTML5. La fonctionnalité de conversion

est disponible par le menu **Média** du logiciel **Convertir/Enregistrer** puis la commande. Le logiciel est téléchargeable à l'adresse www.videolan.org/vlc/.

online-convert

Ce convertisseur en ligne, accessible par l'adresse online-convert.com, offre une bonne rapidité et qualité de conversion. Néanmoins, il est à réserver aux fichiers peu volumineux.

Intégration d'une image SVG

Le langage SVG

SVG est l'acronyme de **Scalable Vector Graphic** qui signifie Graphique Vectoriel Extensible. Seuls les navigateurs récents prennent nativement en charge ce langage, pourtant recommandé par **le W3C depuis 2003**. Ce langage permet de créer des **images vectorielles basées sur le langage XML** et son cortège de balises et d'attributs. Utilisant la même syntaxe à balises que le HTML, le SVG a l'avantage de faire **partie intégrante du DOM** de la page HTML ; ses éléments pouvant donc être ciblés avec du JavaScript.



À savoir

L'acronyme DOM signifie *Document Object Model*, Modèle Objet du Document en français. Le DOM est l'interface entre un langage de programmation et un document XML. Utilisé dans une page HTML, le DOM permet de cibler, de créer, de modifier ou de supprimer une balise de la page.

Les principaux logiciels vectoriels, tels qu'Illustrator ou Inkscape, supportent l'export au format SVG. Le **SVG étant un langage à balises**, il vous est possible de l'apprendre, et ainsi de vous passer de l'utilisation d'un logiciel pour créer des images SVG.

Il existe différentes manières d'intégrer du code SVG dans une page HTML. Pour utiliser la meilleure méthode, il faut distinguer le

fait que le code SVG contienne ou pas du code JavaScript imbriqué dans ses balises, mais aussi de voir s'il est ou non contenu dans un fichier externe. À partir des conditions précédentes, on peut distinguer trois cas d'utilisation du code SVG dans une page HTML :

- si vous souhaitez que le code SVG soit imbriqué parmi le code HTML, vous utiliserez la **balise <svg>**.
- si le code SVG ne contient aucun JavaScript et est encapsulé dans un fichier externe, il peut être considéré comme une image et être appelé par une **balise **.
- si le code SVG est externe et contient du JavaScript, il faudra utiliser une **balise <object>**.

Utilisation de la balise <svg>

HTML

```
<svg width="200" height="200">
<circle cx="100" cy="100" r="50" fill="rgba(255, 255, 0, 0.2)"
stroke="rgb(0, 0, 255)" stroke-width="5" />
</svg>
```

L'imbrication du code SVG dans le code HTML peut être intéressante dans certains cas. Par exemple, le SVG faisant partie du DOM de la page HTML, les textes que vous utilisez dans l'image SVG sont référençables par un moteur de recherche.

Autre exemple, le SVG imbriqué dans le HTML pour un site à destination des mobiles permet d'éviter une requête HTTP vers le serveur, ce qui économise de la batterie.

Utilisation de la balise

HTML

```

```

Si le code SVG ne contient que la description de formes vectorielles sans aucun code JavaScript, il peut être importé comme une **simple image** à l'aide de la balise ``. On peut penser que dans un futur proche, seules les photographies seront dans un format d'image autre que le SVG. Pour l'intégration d'illustrations ou d'images créées à l'origine vectoriellement, comme les logos, il sera plus intéressant d'utiliser le SVG.

Utilisation de la balise <object>

HTML

```
<object data="cercleSVGclic.svg" width="200" height="200"></object>
```

Le fichier contenant le code SVG étant externe et ayant intégré du code JavaScript dans une balise SVG, l'utilisation de la balise <object> est nécessaire pour que **le code reste actif**. Si vous utilisez la balise , les formes vectorielles seront visibles mais le code JavaScript sera désactivé. La balise <object> n'est pas auto-fermante.

- Image de substitution pour les anciens navigateurs

HTML

```
<object data="cercleSVG.svg" width="200" height="200">

</object>
```

Si vous souhaitez utiliser des images SVG, en vous souciant néanmoins des anciens navigateurs, il faudra prévoir une image de substitution à votre image SVG. L'image sera de type GIF, PNG ou JPG et son appel se fera par une **balise imbriquée dans la balise <object>**. Comme pour la vidéo ou l'audio, les anciens navigateurs n'afficheront que le format d'image qu'ils peuvent lire ; dans notre exemple, uniquement la version PNG.

Lorsque l'on maîtrise le SVG...

Pour admirer les possibilités du SVG avec l'utilisation du JavaScript pour l'animation, consultez la page : <http://svg-wow.org/>

Intégration d'un fichier SWF

Le format SWF

SWF signifie **ShockWave Flash**, c'est le format d'un fichier créé à l'aide du **logiciel Flash**. Flash est très populaire pour créer des animations ou des sites dits « full flash », c'est-à-dire exclusivement basés sur la technologie flash et son langage de développement

ActionScript. Son principal inconvénient est qu'il est **gourmand en ressources**, et de ce fait, le lecteur Flash n'a pas été intégré dans les produits nomades d'Apple.

L'intégration d'un fichier SWF est très proche de celle utilisée pour les images SVG, avec l'utilisation possible de différentes balises :

- si vous souhaitez afficher un SWF sans aucune image de substitution, dans le cas où l'internaute ne dispose pas du lecteur Flash, vous pouvez utiliser la **balise <embed>**.
- si vous souhaitez garder une compatibilité avec les supports ne disposant pas du lecteur Flash, en affichant une image de substitution, vous utiliserez la **balise <object>**.

Enfin, des balises ou attributs HTML peuvent être utilisés pour paramétriser l'affichage du fichier SWF, ou pour lui envoyer des informations sous forme de variables appelées « flashvars ».

Utilisation de la balise <embed>

HTML

```
<embed src="flash.swf" width="400" height="400"  
type="application/x-shockwave-flash" />
```

La balise **<embed>** permet d'intégrer un élément dont l'affichage nécessite un lecteur externe ou une extension. C'est le cas, par exemple, des fichiers PDF, des fichiers QuickTime et donc des fichiers SWF. Pour un affichage correct, il est recommandé de définir des dimensions à la balise **<embed>**, car sinon une taille par défaut va être appliquée à l'élément affiché. La spécification du type MIME du fichier n'est pas obligatoire avec la balise **<embed>**.

□ Paramétrage du SWF

HTML

```
<embed src="flash.swf" width="400" height="400"  
wmode="transparent" menu="false"  
flashvars="variable=Message envoyé par les flashvars" />
```

L'affichage peut être paramétré à l'aide d'attributs :

- **wmode** : permet de rendre transparent le fond du SWF.
- **menu** : permet de cacher une partie du menu contextuel.

- ▶ **flashvars** : permet d'envoyer des informations au fichier SWF.
Notez que chaque type de fichiers possède des paramètres qui lui sont spécifiques.

Utilisation de la balise **<object>**

HTML

```
<object data="flash.swf" width="400" height="400"  
type="application/x-shockwave-flash"></object>
```

Comme pour la balise **<embed>**, il est recommandé de définir une dimension à la balise **<object>**, mais contrairement à sa consœur, le **type MIME devient obligatoire** pour une parfaite compatibilité sur tous les navigateurs.

□ Paramétrage du SWF

HTML

```
<object data="flash.swf" width="400" height="400"  
type="application/x-shockwave-flash">  
<param name="wmode" value="transparent" />  
<param name="menu" value="false" />  
<param name="flashvars"  
value="variable=Message envoyé par les flashvars" />  
</object>
```

Avec la balise **<object>**, le paramétrage du fichier incorporé se fait grâce aux balises **<param>** et à ses attributs *name* et *value*. Les balises **<param>** permettent de spécifier un **paramètre unique**.

□ Image de substitution pour les navigateurs ne possédant pas le lecteur Flash

HTML

```
<object data="flash.swf" width="400" height="400"  
type="application/x-shockwave-flash">  
  
</object>
```

Une balise **** imbriquée dans une balise **<object>** permet aux navigateurs ne possédant pas le lecteur Flash d'afficher une **image de substitution**.



Les formulaires

Les formulaires étaient une grande nouveauté du HTML2 sorti en 1994, depuis rien... Ils **ne suivent pas l'évolution du Web** depuis plus d'une dizaine d'années. L'affichage d'un simple calendrier, si courant dans les sites de réservation, est aujourd'hui entièrement dévolu au JavaScript. Le HTML5 change enfin la donne. Outre l'introduction de nouveaux types de champs, de nouveaux attributs permettent de bénéficier d'une **validation des données côté client**, le JavaScript perdant ainsi un domaine de compétences.

Les auteurs tiennent à préciser que les indications sur l'implémentation des formulaires HTML5 se conforment aux versions des navigateurs disponibles au moment de l'écriture de cet ouvrage, c'est-à-dire en juin 2011.

Les nouveaux champs de formulaires

Avec le HTML5, la balise `<input>` voit arriver de nouveaux venus, qui font désormais entrer les formulaires dans le Web 2.0. Néanmoins, la plupart des navigateurs n'offrent pas une expérience utilisateur satisfaisante pour réellement les apprécier, mais ils restent utilisables sous la forme de simples champs de type texte.

Seul le navigateur Opera offre, à ce jour, une réelle interface pour les nouveaux champs de saisie de formulaire. Testez donc vos nouveaux formulaires en remerciant la Norvège...

Les champs de saisie

NOUVEAUX CHAMPS DE SAISIE

Type	Description
email	Saisie d'une adresse mail
tel	Saisie d'un numéro de téléphone
url	Saisie d'une adresse Web
color	Saisie d'une couleur hexadécimale
search	Saisie d'une recherche

□ Le type email

HTML

```
<input type="email" />
```

Qui n'a jamais pesté contre la validation d'un mail avec du JavaScript ?

Ce nouveau type de champ permet de vérifier si la **syntaxe du mail** est correcte, mais non pas que le mail existe. Si ce champ comporte une saisie, il sera **validé par défaut** par les navigateurs Firefox, Opera et Chrome. Sur les mobiles, les touches « arobase » et « point » apparaîtront sur le clavier.

□ Le type tel

HTML

```
<input type="tel" />
```

Ce champ n'est pas validé par défaut par les navigateurs actuels et ne le sera sans doute jamais, vu le nombre de formats téléphoniques mondiaux. Il faudra utiliser une **expression régulière grâce à l'attribut pattern** pour valider un format téléphonique national. Son intérêt est plus évident sur les mobiles puisqu'il permet d'afficher un clavier ne comportant que des chiffres.

□ Le type url

HTML

```
<input type="url" />
```

Ce champ attend la saisie d'une adresse Web valide syntaxiquement. S'il comporte une saisie, il sera **validé par défaut** par les navigateurs Firefox, Opera et Chrome mais dans des formes différentes :

- Firefox attend les caractères *http:* en début de saisie.
- Opera ajoute lui-même les caractères *http://* en début de saisie lors de l'envoi du formulaire.
- Chrome attend au minimum les caractères *http://* en début de saisie.

Sur les mobiles, les touches « point », « barre oblique » et « .com » apparaîtront en supplément sur le clavier.

□ Le type color

HTML

```
| <input type="color" />
```

Le navigateur Opera est le seul, à ce jour, à permettre de sélectionner une couleur *via une pipette*. Sur les autres navigateurs, vous devrez saisir le code hexadécimal de la couleur dans un champ de saisie sans aucune validation.

□ Le type search

HTML

```
| <input type="search" />
```

Le type *search* sert à indiquer à l'internaute qu'il saisit dans un champ de recherche. Pour différencier ce champ d'un simple champ de saisie, les navigateurs l'ont signalé visuellement par une **forme particulière ou des icônes**. Par exemple, sur Safari, Opera et Chrome dans leurs versions Mac OS X, il apparaît avec des bords arrondis. Sur Chrome et Safari dans leurs versions Windows, seule une croix apparaît à droite du champ pour effacer la recherche.

Les champs numériques

Les champs numériques des formulaires HTML5 apparaîtront correctement dans la plupart des navigateurs. Dans ceux ne les

ayant pas encore implémentés, ils apparaîtront comme des champs de saisie.

Le type number

HTML

```
<input type="number" />
```

Ce type de champ permet d'afficher un **champ de saisie avec des flèches** à sa droite pour incrémenter ou décrémenter un chiffre. Cette interface n'est disponible, à ce jour, que sur les navigateurs Opera, Chrome et Safari, mais aucun d'entre eux ne valide la saisie d'un chiffre dans ce champ.

Le type range

HTML

```
<input type="range" />
```

Le type *range* permet, à l'aide d'un **curseur**, de sélectionner un chiffre parmi une plage de valeurs. Si vous ne spécifiez pas de valeur minimale et maximale, la plage de valeurs ira de 0 à 100.

Les champs temporels

Le HTML5 introduit une série de champs servants à récupérer **une date, un mois ou une heure précise**. Seuls les navigateurs Opera et Safari ont implémenté une interface utilisateur complète pour ces champs.

Le tableau suivant récapitule tous ces nouveaux champs temporels avec la saisie – au format ISO – attendue par le navigateur.

Type	Description	Exemple de saisie
date	Saisie d'une date complète	2012-02-28
datetime	Saisie d'une date et d'une heure en Temps Universel Coordonné (TUC)	2012-02-28T17:04:07Z
datetime-local	Saisie d'une date et d'une heure selon le fuseau horaire de l'internaute	2012-02-28T18:04:07
time	Saisie d'une heure	18:04:07

Type	Description	Exemple de saisie
month	Saisie d'un mois	2012-02
week	Saisie d'une semaine	2012-W09

□ Le type date

HTML

```
<input type="date" />
```

Enfin un **calendrier natif** en HTML ! Le type *date* permet la sélection d'un jour, d'un mois et d'une année.

□ Les types datetime et datetime-local

HTML

```
<input type="datetime" />
<input type="datetime-local" />
```

Ce type de champ permet de sélectionner une date accompagnée d'une heure.



À savoir

Le Temps Universel Coordonnée (UTC en anglais) se base sur l'heure GMT (*Greenwich Mean Time*) et l'heure Z (Zulu). Le TUC est basé sur des calculs prenant en compte la rotation terrestre, alors que l'heure GMT se base sur une durée moyenne de rotation ; de plus, il commence à minuit alors que le temps GMT débute à midi. Il correspond désormais à la norme internationale.

Le type *datetime* se base sur le temps TUC dont le premier fuseau horaire est centré sur le Méridien de Greenwich.

Le type *datetime-local* se base sur l'heure locale de l'internaute en prenant donc en compte son fuseau horaire.

Lors de la saisie ou de la récupération de la saisie, le caractère « T » sert de séparateur entre la date et l'heure. Concernant le type *datetime*, le caractère « Z » indique l'utilisation de l'heure Zulu.

□ Le type time

HTML

```
<input type="time" />
```

La sélection d'une heure se fait avec le type *time*. L'heure est composée de trois séries de chiffres représentant les heures, les minutes et les secondes.

□ Les types month et week

HTML

```
<input type="month" />  
<input type="week" />
```

Les types *month* et *week* permettent respectivement de sélectionner un mois ou une semaine. Concernant le type *week*, lors de la saisie ou de la récupération de la saisie, le caractère « W » sert de séparateur entre l'année et le numéro de la semaine.

Les nouvelles balises de formulaires

Conjointement aux nouveaux champs de formulaires, de nouvelles balises apparaissent avec le HTML5 permettant principalement d'améliorer l'ergonomie des formulaires.

La balise <datalist>

HTML

```
Club de football favori :  
<input type="text" id="clubs" list="listeClubs" />  
<datalist id="listeClubs">  
  <option value="Paris Saint-Germain" label="psg">  
  <option value="Marseille" label="om">  
  <option value="Lyon" label="ol">  
  <option value="Losc" label="lille">  
</datalist>
```

La balise **<datalist>** affiche un ensemble de propositions de saisie. Cette balise se comporte comme une liste déroulante à la différence qu'elle est dépendante d'un champ de saisie. D'après

nos tests avec le navigateur Opera, il est possible de lier la balise `<datalist>` aux types de saisie suivants :

```
text  
tel  
url  
color  
search  
range.
```

La balise `<datalist>` doit comporter un attribut **id** précisant son nom unique, et permettant au champ qui en dépend de la cibler avec l'attribut **list**. Cette balise doit contenir des balises `<option>` précisant chacune une **proposition possible**.

La validation actuelle du HTML5 n'autorise pas que les balises `<option>` soient auto-fermantes, ou même comportent une balise de fermeture. De fait, une balise `<option>` doit comporter deux attributs :

- **attribut value** : spécifie la valeur de la proposition ; valeur qui sera par la suite récupérée par une page serveur.
- **attribut label** : permet de spécifier le texte lié à la proposition.

Il est à noter qu'il est possible de se passer de l'attribut **label** si sa valeur est identique à celle de l'attribut **value**.

Grâce à la balise `<datalist>`, il est désormais simple de simuler la célèbre fonctionnalité d'**auto-complétion** présente dans la plupart des moteurs de recherche actuels.

La balise `<progress>`

HTML

```
<progress value="0.95">95%</progress>
```

La balise `<progress>` permet d'afficher une barre de progression. Elle doit être utilisée pour informer l'internaute de l'état de progression d'un élément ou d'une tâche, comme la position de la tête de lecture dans une vidéo ou la progression d'un transfert de fichier. Si vous ne spécifiez pas de valeur minimale et maximale, la plage de valeurs ira de 0 à 1. À ce jour, Internet Explorer et Safari n'ont pas implémenté cette balise.

La balise <meter>

HTML

```
<progress value="0.95">95%</progress>
```

La balise **<meter>** permet d'afficher une gauge. À la différence de la balise **<progress>**, la balise **<meter>** sert à indiquer à l'internaute une valeur numérique fixe à un moment donné, par exemple le taux de remplissage d'une salle de concert ou le temps restant pour une enchère. Si vous ne spécifiez pas de valeur minimale et maximale, la plage de valeurs ira de 0 à 1. À ce jour, seuls Opera et Chrome ont implémenté cette balise.

Les nouveaux attributs

Poursuivant sa philosophie d'amélioration de l'ergonomie des sites Web, le HTML5 permet désormais d'attribuer des comportements spécifiques aux champs de formulaires. Ces comportements nécessitaient auparavant l'intervention du JavaScript.

les attributs des champs numériques et temporels

□ Les attributs value, min et max

HTML

```
<input type="number" value="30" min="30" max="100" />
<input type="range" value="1" min="1" max="10" />
<meter value="200" min="0" max="1000">200</meter>
<input type="datetime-local" value="2012-05-21T12:00" min="2012-05-21T12:00" max="2012-05-26T16:00" />
```

Parmi les champs numériques que nous avons découverts, nous ne leur avons jamais spécifié de valeurs limites. Ainsi, par défaut, certains champs numériques ont pour base 1 ou 100, alors que les champs temporels n'ont pas de limites définies.

Les attributs **min** et **max** vous permettent de paramétriser les valeurs minimale et maximale des champs numériques et temporels. L'intérêt est évident pour les champs de type **number** ou **range** mais aussi très intéressant pour les champs temporels, car vous avez la possibilité de définir une plage temporelle.

L'attribut **value**, quant à lui, vous permet d'attribuer une valeur initiale à un champ.

□ L'attribut step

HTML

```
<input type="number" value="30" min="30" max="100" step="5" />  
<input type="range" value="1" min="1" max="10" step="1" />
```

L'attribut **step** permet de spécifier le pas d'incrémentation d'un champ numérique.

Les attributs d'obligation et de validation des champs de saisie

□ L'attribut required

HTML

```
<input type="email" required />  
<p>Couleur préférée : <input type="color" required /></p>  
<p>Semaine : <input type="week" required /></p>
```

Un petit pas pour le HTML, un grand pas pour les non-développeurs !

L'attribut **required** spécifie que le champ est obligatoire, le navigateur s'occupant même de le rappeler à l'internaute au travers de l'affichage d'un message. Ce simple attribut permet d'économiser des dizaines de lignes de code JavaScript. Néanmoins, il faut préciser qu'il ne valide pas la saisie ; il ne fait que rendre obligatoire cette dernière, à l'exception des champs de type email et url. À ce jour, Internet Explorer et Safari n'ont pas ou partiellement implémenté cet attribut.

□ L'attribut pattern

HTML

```
Code postal : <input type="text" pattern="[0-9]{5}" />  
Mobile : <input type="text" pattern="0[6-7][0-9]{8}" />
```

Complémentaire de l'attribut **required**, l'attribut **pattern** permet, quant à lui, de valider la saisie par l'intermédiaire des expressions régulières. Ces dernières décrivent un **modèle syntaxique** pour

une donnée et ne sont pas liées à un langage particulier. Vous pouvez donc utiliser des expressions régulières créées pour d'autres langages comme le PHP ou le JavaScript. Même si un livre entier est nécessaire pour les apprendre, quelques fragments permettent d'en comprendre les bases :

- [0-9] : saisie d'un chiffre de 0 à 9.
- [a-z] : saisie d'une minuscule de a à z.
- [A-Z] : saisie d'une capitale de A à Z.
- {nombre} : nombre exact de caractères attendus.
- {nombre minimal, nombre maximal} : une plage de nombre de caractères attendus.

Visitez le site **Expreg.com**

Rendez-vous à l'adresse <http://www.expreg.com/> pour trouver et/ou apprendre le langage des expressions régulières.

Les exemples présentés précédemment permettent respectivement la saisie d'un code postal français composé de 5 chiffres, et d'un numéro de mobile français commençant par 06 ou 07 et suivi de 8 chiffres.



Attention !

Les attributs required et pattern, même s'ils sont d'une grande aide pour vérifier en amont la saisie d'un internaute, ne supprimeront jamais la validation des données côté serveur.

Rappelons qu'un grand nombre d'attaques sur les sites Web se font au travers des champs de saisie des formulaires.

Les attributs ergonomiques des champs de saisie

□ L'attribut autofocus

HTML

■ Nom : <input type="text" autofocus />

Aujourd'hui, lorsque vous êtes sur un moteur de recherche, le curseur se place automatiquement dans le champ de recherche, cela vous évite de sélectionner le champ alors qu'il y a 99,99 % de chance pour que vous le fassiez dès votre arrivée sur le site.

Cette fonctionnalité de **focalisation automatique** sur un champ de saisie est désormais accessible avec l'attribut **autofocus**. Son utilisation doit néanmoins se faire sous certaines conditions :

- ▶ il est généralement utilisé sur le premier champ du formulaire.
- ▶ le formulaire doit être le sujet principal de la page.

□ L'attribut placeholder

HTML

■ Site Web : <input type="url" placeholder="par exemple www." />

L'attribut **placeholder** permet d'afficher un **message informatif** dans le champ de saisie. Ce message aide l'internaute dans sa saisie et évite les erreurs de validation du formulaire.

□ L'attribut autocomplete

HTML

■ Recherche : <input type="search" autocomplete="off" />

L'attribut **autocomplete** permet de définir l'**auto-complétion**, c'est-à-dire le remplissage automatique du champ de saisie grâce à vos saisies précédentes. Deux valeurs sont possibles pour cet attribut :

- ▶ **valeur on** : l'auto-complétion est active, c'est la valeur par défaut si l'attribut n'est pas spécifié.
- ▶ **valeur off** : l'auto-complétion est inactive.

Pour la sécurité de vos internautes, vous pourriez, par exemple, définir que le remplissage automatique du champ de saisie pour le numéro de sa carte bleue est inactif.



7

Création graphique

Le HTML5 introduit la **balise <canvas>** qui permet la création d'images dynamiques matricielles (ou *bitmap* en anglais) directement dans le navigateur, sans passer par un plug-in ou un langage serveur. La création dynamique d'images étant désormais possible, un ensemble de possibilités apparaissent comme la **création d'application de dessin, de jeu ou encore l'animation**.

Néanmoins, si **<canvas>** est bel et bien du HTML5, les méthodes de dessin sont l'affaire du JavaScript. Ne vous attendez donc pas à des balises ou à des attributs mais plutôt à des méthodes ou à des objets JavaScript. Nous supposons, par conséquent, que vous connaissez les bases du JavaScript ou de tout autre langage de programmation.

De plus, nous allons utiliser la **bibliothèque jQuery**, disponible à l'adresse <http://jquery.com/> pour le code JavaScript.

Le canevas

La balise <canvas>

HTML

```
<!DOCTYPE HTML>
<html lang="fr">
<head>
<meta charset="utf-8" />
```

```
<meta name="viewport" content="width=device-width, initial-  
scale=1.0, user-scalable=no" />  
<title>Canevas</title>  
<!-- JQuery -->  
<script type="text/javascript" src="jquery.js"></script>  
<!-- JavaScript -->  
<script type="text/javascript"></script>  
</head>  
<body>  
<canvas width="400" height="400" id="canevas">  
</canvas>  
</body>  
</html>
```

Notre page HTML importe la bibliothèque jQuery, puis utilise une **balise <script>** qui nous permettra d'écrire le code JavaScript nécessaire. Quant à la balise **<canvas>**, du point de vue HTML, elle est assez simpliste dans la mesure où ses méthodes de dessin sont dévolues au JavaScript. Les attributs, tous optionnels, de la balise sont :

- **width** : largeur en pixels ;
- **height** : hauteur en pixels ;
- **id** : identifiant unique du canevas.

Si vous ne spécifiez pas la largeur et la hauteur du canevas, la balise prendra les dimensions de 300 pixels de largeur sur 150 pixels de hauteur.

Un canevas est par défaut accompagné d'un **fond transparent**, ce qui rend la balise invisible si aucun contour ou fond n'est ajouté.

Important : en termes de CSS, la balise **<canvas>** est de **type en ligne** ; elle ne provoque donc pas de retour à la ligne comme avec les balises de **type bloc**. Cela s'avère très pratique, car cela permet, par exemple, de dessiner une puce pour illustrer une énumération.

Compatibilité des navigateurs

HTML

```
<canvas width="400" height="400" id="canevas">  
<p>Votre navigateur n'aime pas le dessin.</p>  
</canvas>
```

Comme avec toutes les nouvelles balises HTML5, il est nécessaire que l'internaute dispose d'un navigateur récent.

La balise `<canvas>` est avant tout destinée à servir d'interface pour l'affichage du dessin ; elle ne devrait donc pas inclure de contenu, uniquement du code. C'est une **balise sans contenu**. Néanmoins, il est possible d'y ajouter un paragraphe qui informera l'internaute du non-support de la balise par son navigateur.

Le type de rendu

JAVASCRIPT

```
<!-- JQuery -->
<script type="text/javascript" src="jquery.js"></script>
<!-- JavaScript -->
<script type="text/javascript">
// attente du chargement complet de la page HTML
$(document).ready(init);
// initialisation
function init(e){
    var contexte2D = $("#canevas")[0].getContext("2d");
}
</script>
```

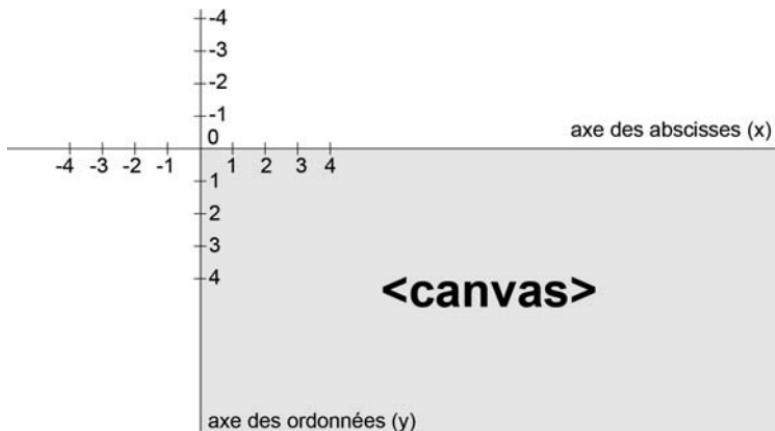
La balise `<canvas>` n'étant donc que l'interface d'affichage, il est ensuite impératif de spécifier **le type de rendu** que l'on souhaite.

La méthode **getContext** permet d'accéder à toutes les méthodes de dessin.

À l'heure actuelle, **seul le dessin en 2 dimensions est compatible** avec tous les navigateurs récents, la troisième dimension (3D) est en cours de développement. Un dessin en 2D est caractérisé par son fond et son contour.

Les coordonnées du canevas

Le canevas est un **plan cartésien**, mais l'axe des ordonnées est inversé, c'est-à-dire que les valeurs positives sont au-dessous de l'axe des abscisses. Le **point origine** du canevas est le coin haut gauche de la balise.



Dessiner des formes simples

Formes rectangulaires

JAVASCRIPT

```
function init(e){  
    var contexte2D = $("#canevas")[0].getContext("2d");  
    // rectangle avec fond  
    contexte2D.fillRect(25, 25, 200, 100);  
    // rectangle avec contour  
    contexte2D.strokeRect(25, 150, 50, 150);  
}
```

Pour dessiner une forme rectangulaire, trois méthodes s'offrent à vous :

- **rect** : dessiner un rectangle sans couleur de fond ni contour.
- **fillRect** : dessiner un rectangle avec une couleur de fond.
- **strokeRect** : dessiner un rectangle avec uniquement un contour.

Ces trois méthodes nécessitent des paramètres identiques, dont les valeurs sont exprimées en pixels :

- ▶ la position du rectangle en abscisse (x) ;
- ▶ la position du rectangle en ordonnée (y) ;
- ▶ la largeur du rectangle ;
- ▶ la hauteur du rectangle.

Dans notre exemple, le rectangle plein est dessiné à partir de la position 25 en x et 25 en y, puis possède des dimensions de 200 pixels en largeur et 100 pixels en hauteur. Concernant le contour du second rectangle, sa position est de 25 pixels en x, 150 pixels en y, et ses dimensions sont de 50 pixels en largeur et 150 pixels en hauteur. Par défaut, le fond et le contour sont de couleur noire.

Tracés

JAVASCRIPT

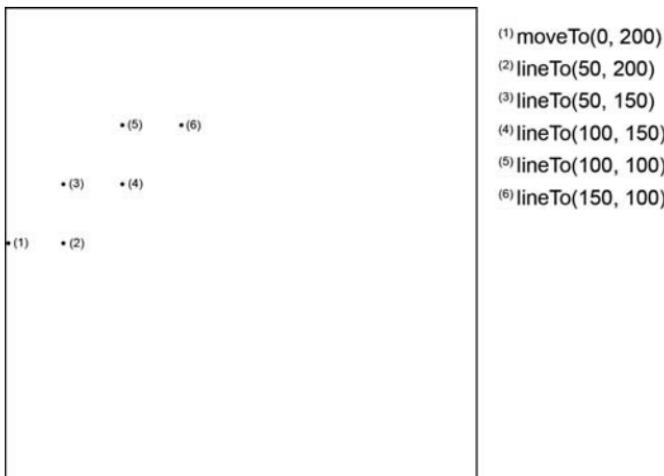
```
// tracé en escaliers
contexte2D.beginPath();
contexte2D.moveTo(0, 200);
contexte2D.lineTo(50, 200);
contexte2D.lineTo(50, 150);
contexte2D.lineTo(100, 150);
contexte2D.lineTo(100, 100);
contexte2D.lineTo(150, 100);
contexte2D.stroke();
contexte2D.closePath();
```

L'exemple présenté ci-dessus permet de dessiner un escalier. Pour créer des tracés, il faut utiliser plusieurs méthodes dans un ordre strict :

- ▶ **beginPath** : permet de déclarer la création d'un tracé.
- ▶ **moveTo** : permet d'indiquer le point initial d'un tracé.
- ▶ **lineTo** : permet de créer un segment en indiquant les coordonnées de l'extrémité de celui-ci.
- ▶ **stroke** : permet de faire le rendu du contour.
- ▶ **closePath** : permet de fermer un tracé.

Important : l'ordre des instructions est primordial. Si par exemple, vous intervertissez les méthodes `stroke` et `closePath`, cela signifie que vous souhaitez fermer le tracé puis faire le rendu du contour. Le tracé sera alors fermé par un dernier segment créé

automatiquement par la méthode stroke. Dans notre exemple, il est composé de cinq segments dont les coordonnées des extrémités sont présentées ci-dessous.



Polygones

JAVASCRIPT

```
// tracé fermé ou polygone
contexte2D.beginPath();
contexte2D.moveTo(50, 150);
contexte2D.lineTo(80, 150);
contexte2D.lineTo(80, 135);
contexte2D.lineTo(105, 155);
contexte2D.lineTo(80, 175);
contexte2D.lineTo(80, 160);
contexte2D.lineTo(50, 160);
contexte2D.lineTo(50, 150);
contexte2D.closePath();
contexte2D.fill();
```

L'exemple présenté ci-dessus permet de dessiner une flèche. Un polygone n'est qu'un tracé, mais il a la particularité d'être fermé. Pour fermer un tracé, il suffit de créer des segments dont le point initial coïncide avec le point final du dessin. Chaque appel à la méthode **lineTo** déclenche le dessin d'un côté du polygone. Par

exemple, si vous souhaitez créer un hexagone, vous utiliserez à six reprises la méthode **lineTo**. Une fois le polygone dessiné, il faut effectuer le rendu du tracé, en utilisant soit la méthode **stroke** pour ajouter un contour, soit la méthode **fill** pour ajouter une couleur de fond, ou les deux conjointement.

Dessiner des formes complexes

Arc de cercle par utilisation d'angles

JAVASCRIPT

```
// cercle
contexte2D.beginPath();
contexte2D.arc(200, 200, 100, 0, Math.PI * 2);
contexte2D.fill();
contexte2D.closePath();
// arc de cercle
contexte2D.beginPath();
contexte2D.arc(200, 200, 120, 0, Math.PI, true);
contexte2D.stroke();
contexte2D.closePath();
```

La méthode **arc** permet de dessiner d'une part un arc de cercle en spécifiant un angle initial et un angle final, mais également un cercle complet. Ses paramètres sont les suivants :

- la position du centre de l'arc en abscisse (x) ;
- la position du centre de l'arc en ordonnée (y) ;
- le rayon de l'arc ;
- l'angle initial en radians ;
- l'angle final en radians ;
- sens de rotation du tracé de l'arc.

Concernant le sens de rotation du tracé de l'arc, c'est un **paramètre optionnel** qui nécessite une **valeur booléenne** :

- **true** : permet de dessiner dans le sens inverse des aiguilles d'une montre.

- ▶ **false** : permet de dessiner dans le sens des aiguilles d'une montre, c'est la valeur par défaut.



À savoir

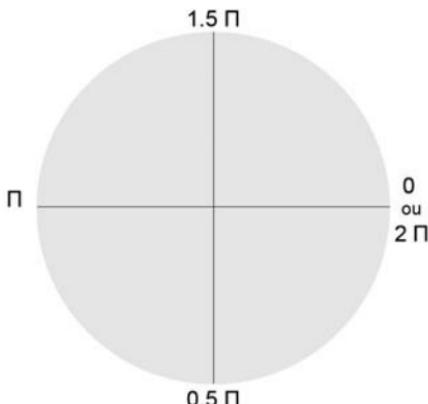
Si vous souhaitez exprimer les angles en degrés plutôt qu'en radians (plus simple à imaginer surtout quand les cours de géométrie des années de collège remontent à longtemps...), voici la formule de conversion : $(\text{angle en degrés} \times \pi) / 180$.

Autre solution : saisissez « 1 degré en radian » dans le champ de recherche de Google ; vous utilisez ainsi l'outil de conversion du moteur de recherche, qui vous affichera alors gracieusement le résultat de 0,0174532925 radian.

Dans notre exemple, pour dessiner notre cercle, nous avons utilisé les angles en radians suivants :

- ▶ 0π correspond à 0° , c'est l'angle initial du cercle.
- ▶ 2π correspondent à 360° , c'est l'angle final du cercle.

Le schéma ci-dessous permet de visualiser les angles en radians.



Convertir les degrés en radians**JAVASCRIPT**

```
// arc de cercle avec les angles exprimés en degrés
contexte2D.beginPath();
contexte2D.arc(200, 200, 140, degresEnRadians(90),
degresEnRadians(270));
contexte2D.stroke();
contexte2D.closePath();

// degrés en radians
function degresEnRadians(angleDegres){
    return (angleDegres * Math.PI) / 180;
}
```

Pour faciliter la création d'arc de cercle, il est conseillé de créer une fonction JavaScript qui permet la conversion des degrés en radians. Il suffira ensuite de l'utiliser comme paramètre de la méthode **arc**.

Arc de cercle par utilisation de tangentes

JAVASCRIPT

```
// rectangle avec bords arrondis
contexte2D.beginPath();
contexte2D.moveTo(50, 50);
contexte2D.arcTo(200, 50, 200, 100, 20);
contexte2D.lineTo(200, 200);
contexte2D.arcTo(70, 180, 60, 130, 20);
contexte2D.lineTo(50, 50);
contexte2D.fill();
contexte2D.closePath();
```

Alors que la méthode **arc** nécessite des angles en radians pour créer un arc de cercle, la méthode **arcTo** utilise les coordonnées de deux points qui vont servir de repères pour créer les tangentes. Les coordonnées de ces deux tangentes vont créer **deux lignes virtuelles à partir du point précédent du tracé**, et ainsi produire l'arc de cercle. Les tangentes n'indiquant que des directions, **une infinité de valeurs sont possibles** pour les coordonnées des points de repère, à partir du moment où l'angle d'intersection des tangentes reste identique.

Par exemple, avec le dessin précédent, si vous remplacez l'appel des deux méthodes **arcTo** par les lignes suivantes, le résultat visuel sera identique :

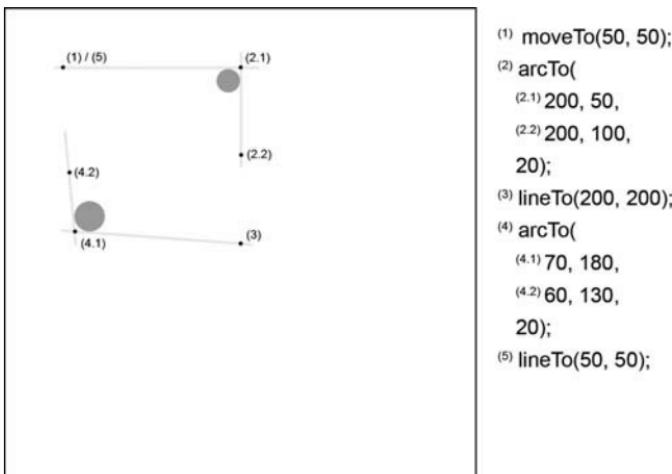
JAVASCRIPT

```
contexte2D.arcTo(200, 50, 200, 200, 20);  
contexte2D.arcTo(70, 180, 50, 50, 20);
```

La méthode **arcTo** est recommandée lorsque vous souhaitez dessiner un tracé avec un arc de cercle sur un segment. Les paramètres de la méthode sont :

- ▶ la position x du point de repère de la première tangente ;
- ▶ la position y du point de repère de la première tangente ;
- ▶ la position x du point de repère de la seconde tangente ;
- ▶ la position y du point de repère de la seconde tangente ;
- ▶ le rayon de l'arc de cercle.

Dans notre exemple, nous avons dessiné un rectangle avec deux angles arrondis opposés dont les coordonnées des points de repère sont indiquées ci-dessous.



Courbes quadratiques

JAVASCRIPT

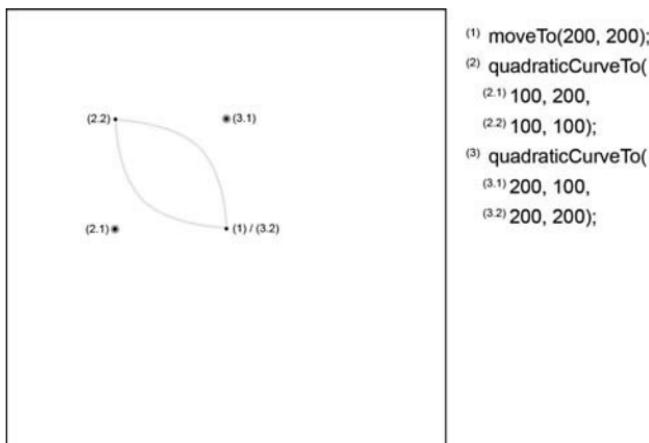
```
// courbes quadratiques  
contexte2D.beginPath();  
contexte2D.moveTo(200, 200);  
contexte2D.quadraticCurveTo(100, 200, 100, 100);  
contexte2D.quadraticCurveTo(200, 100, 200, 200);  
contexte2D.fill();  
contexte2D.closePath();
```

Les courbes quadratiques sont caractérisées par leur **forme parabolique**. On peut imaginer une courbe quadratique comme une ligne qui serait déformée ou bien attirée par un point agissant tel un aimant sur un fil de fer souple. Cette attraction, selon la position de l'aimant, modifie la forme de la courbe.

La méthode **quadraticCurveTo** s'utilise avec les paramètres suivants :

- la position du point aimant de la courbe en x ;
- la position du point aimant de la courbe en y ;
- la position du point final de la courbe en x ;
- la position du point final de la courbe en y.

Notre exemple permet de dessiner deux courbes symétriques se rejoignant au même point.



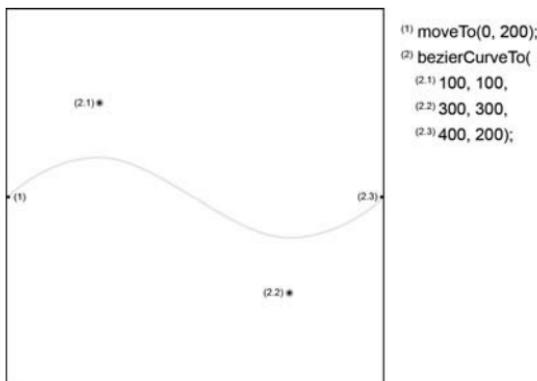
Courbes de Bézier

JAVASCRIPT

```
// courbes de Bézier
contexte2D.beginPath();
contexte2D.moveTo(0, 200);
contexte2D.bezierCurveTo(100, 100, 300, 300, 400, 200);
contexte2D.stroke();
contexte2D.closePath();
```

À la différence des courbes quadratiques, les courbes de Bézier se caractérisent par le fait que leur tracé peut comporter des **courbes dont les formes sont inversées**. Elles sont donc très proches dans leur construction des courbes quadratiques, mais elles possèdent un point aimant supplémentaire. Pour les dessiner, il faut utiliser la méthode **bezierCurveTo** avec six paramètres :

- ▶ la position x du point du premier point aimant de la courbe ;
- ▶ la position y du point du premier point aimant de la courbe ;
- ▶ la position x du point du second point aimant de la courbe ;
- ▶ la position y du second point aimant de la courbe ;
- ▶ la position x du point final de la courbe ;
- ▶ la position y du point final de la courbe.



Le schéma ci-dessus permet de visualiser la position de chaque point créé avec la méthode **bezierCurveTo**.

Styliser les contours et les formes

Jusqu'ici les fonds et les contours de nos formes étaient de couleur noire ; nous allons les égayer par l'ajout de **couleurs unies ou de dégradés**, mais également en réglant **le rendu et l'épaisseur des contours**.

Style des contours

JAVASCRIPT

```
contexte2D.beginPath();
contexte2D.lineWidth = 20;
contexte2D.lineCap = "square";
contexte2D.lineJoin = "bevel";
contexte2D.moveTo(100, 100);
contexte2D.lineTo(150, 50);
contexte2D.lineTo(200, 100);
contexte2D.lineTo(250, 50);
contexte2D.lineTo(300, 100);
contexte2D.stroke();
contexte2D.closePath();
```

□ Épaisseur des contours

La propriété *lineWidth* du contexte du dessin permet de définir l'épaisseur en pixels du contour d'une forme.

□ Aspect de l'extrémité des segments d'un contour

JAVASCRIPT

```
contexte2D.lineCap = "butt";
contexte2D.lineCap = "round";
```

Il est possible, comme dans la majorité des logiciels graphiques tels qu'illustrator, de définir l'aspect de l'extrémité des segments des contours, grâce à la propriété *lineCap* du contexte de dessin. Cette propriété accepte trois valeurs possibles :

- **valeur « butt »** : l'extrémité est carrée et le trait se termine à fin du tracé, c'est la valeur par défaut.
- **valeur « square »** : l'extrémité est carrée et le trait est prolongé au-delà de la fin du tracé.

- **valeur « round »** : l'extrémité est arrondie et le trait est prolongé au-delà de la fin du tracé.

- Aspect des sommets des segments d'un contour

JAVASCRIPT

```
contexte2D.lineJoin = "miter";
contexte2D.lineJoin = "round";
```

Enfin, la propriété *lineJoin* permet de spécifier l'aspect des sommets des segments d'un contour, lorsque le tracé du contour comporte des angles. Les valeurs possibles pour *lineJoin* sont :

- **valeur bevel** : le sommet sera biseauté ;
- **valeur miter** : le sommet sera pointu ;
- **valeur round** : le sommet sera arrondi.

Ajouter des couleurs

- Utiliser une couleur unie

JAVASCRIPT

```
contexte2D.beginPath();
contexte2D.lineWidth = 5;
contexte2D.fillStyle = "#00f";
contexte2D.strokeStyle = "hsla(0, 100%, 50%, 0.75)";
contexte2D.arc(30, 200, 20, 0, Math.PI * 2);
contexte2D.fill();
contexte2D.stroke();
contexte2D.closePath();
```

Les propriétés *fillStyle* et *strokeStyle* permettent respectivement de colorier le fond et le contour d'une forme. Elles acceptent les déclarations de couleurs CSS, dans toutes les formes ou unités :

- **nom d'une couleur Web** : la palette Web comprend 216 couleurs et chacune possède son petit nom, du classique *green* au plus improbable *cornflowerblue*.
- **couleur hexadécimale** : la couleur est décrite sous forme de six caractères précédés d'un dièse ; comme par exemple #OOBBFF pour un bleu ciel. Notez que la notation courte est acceptée ; ainsi, nous aurions pu écrire #OBF au lieu de #OOBBFF, mais le résultat aurait été identique.

- ▶ **couleur en RVB** : la couleur est décrite par ses valeurs Rouge, Verte et Bleu ; ainsi, un bleu ciel en RVB sera noté : `rgb(0, 187, 255)`.
- ▶ **couleur en RVB transparente** : nouveauté des CSS3, la couleur peut désormais être transparente grâce à la méthode `rgba`, le « `a` » final signifie « alpha » pour couche alpha. la valeur de l'alpha est en base 1 et accepte deux décimales après la virgule ; par exemple si l'on souhaite ce bleu ciel avec une transparence de 25 %, nous inscrirons : `rgba(0, 187, 255, 0.25)`.
- ▶ **couleur en TSL** : la couleur est décrite par sa Teinte, et ses valeurs de Saturation et de Luminosité en pourcentage. Par exemple, la valeur TSL du même bleu ciel est `hsl(196, 50 %, 100 %)`.
- ▶ **couleur en TSL transparente** : identique à la méthode `rgba` mais en utilisant cette fois-ci la méthode `hsla` ; par exemple `hsla(196, 50 %, 100 %, 0.25)` pour notre bleu ciel.

□ Utiliser un dégradé linéaire

JAVASCRIPT

```
// création du dégradé linéaire pour le fond
var degLineaireFond = contexte2D.createLinearGradient(60, 150, 60,
250);
degLineaireFond.addColorStop(0, "rgb(255, 255, 255)");
degLineaireFond.addColorStop(1, "rgb(255, 0, 0)");
// dégradé linéaire pour le contour
var degLineaireContour = contexte2D.createLinearGradient(60, 150,
110, 250);
degLineaireContour.addColorStop(0, "rgb(0, 0, 255)");
degLineaireContour.addColorStop(1, "rgb(0, 255, 0)");
// création de la forme
contexte2D.beginPath();
contexte2D.lineWidth = 10;
contexte2D.fillStyle = degLineaireFond;
contexte2D.strokeStyle = degLineaireContour;
contexte2D.arc(110, 200, 50, 0, Math.PI * 2);
contexte2D.fill();
contexte2D.stroke();
contexte2D.closePath();
```

La création d'un dégradé linéaire passe par une **variable JavaScript** qui permet d'indiquer, d'une part, la surface recouverte par le dégradé, et d'autre part, les couleurs composants le dégradé.

La méthode **createLinearGradient** permet de déclarer la création d'un dégradé et nécessite quatre paramètres :

- ▶ la position x du point initial du dégradé ;
- ▶ la position y du point initial du dégradé ;
- ▶ la position x du point final du dégradé ;
- ▶ la position y du point final du dégradé.

La **forme d'un dégradé sera donc toujours rectangulaire**, ce qui peut parfois être gênant visuellement lorsque vous souhaitez remplir un cercle, il se peut que des couleurs apparaissent peu ou pas du tout. Il faut donc que le dégradé soit positionné au même endroit que la forme, et que sa surface soit suffisante pour superposer la totalité de la forme. De plus, la position x/y du point final du dégradé indique également la **direction du dégradé** par l'angle formé avec le point initial ainsi que ses dimensions.

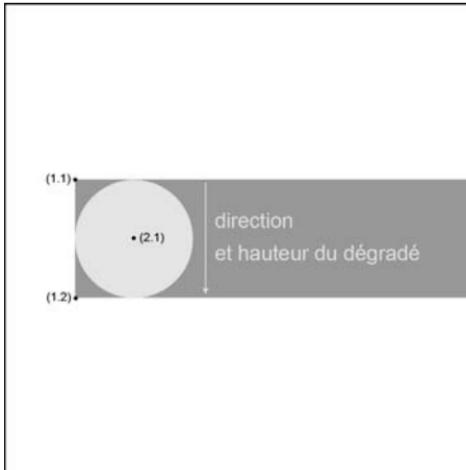
La seconde méthode nécessaire à la création d'un dégradé est **addColorStop** qui permet d'ajouter une couleur au dégradé. Ses deux paramètres sont :

- ▶ la position de la couleur, avec une **valeur décimale de 0 à 1** ;
- ▶ la couleur à ajouter.

Un dégradé doit être imaginé comme une droite dont les valeurs 0 et 1 représentent chacune une extrémité du dégradé ; par exemple, si vous souhaitez intégrer une couleur au milieu du dégradé, sa position sera de 0.5, aux trois-quarts du dégradé, sa position sera 0.75.

Enfin, pour appliquer un dégradé soit sur le fond, soit sur le contour d'une forme, il faut utiliser les propriétés *fillStyle* et *strokeStyle* avec le nom de la variable JavaScript créée.

Dans notre exemple, le dégradé linéaire du fond est vertical car les positions x du point initial et final sont identiques, et la hauteur du dégradé est de 100 pixels qui est la différence entre les positions y des points. Pour le contour, le dégradé linéaire est orienté à 45° car la position x du point initial et la position x du point final sont décalés (60 et 110), et la hauteur du dégradé est de 100 pixels.



(1) createLinearGradient(

(1.1) 60, 150,

(1.2) 60, 250);

(2) arc(

(2.1) 110, 200,

50,

0, Math.PI * 2);

□ Utiliser un dégradé radial

JAVASCRIPT

```
// création du dégradé radial
var degRadial = contexte2D.createRadialGradient(340, 200, 0, 340,
200, 80);
degRadial.addColorStop(0, "rgb(0, 0, 255)");
degRadial.addColorStop(0.2, "rgba(0, 255, 255, 0)");
degRadial.addColorStop(0.3, "rgb(0, 0, 255)");
degRadial.addColorStop(0.5, "rgba(0, 255, 0, 0)");
// création de la forme
contexte2D.beginPath();
contexte2D.fillStyle = degRadial;
contexte2D.arc(340, 200, 40, 0, Math.PI * 2);
contexte2D.fill();
contexte2D.closePath();
```

La création d'un dégradé radial se fait avec la méthode **createRadialGradient** et six paramètres :

- ▶ la position du centre du cercle intérieur en x ;
- ▶ la position du centre du cercle intérieur en y ;
- ▶ le rayon du cercle intérieur ;
- ▶ la position du centre du cercle extérieur en x ;

- ▶ la position du centre du cercle extérieur en y ;
- ▶ le rayon du cercle extérieur.

Il est souvent préférable que les centres des cercles intérieur et extérieur coïncident afin d'obtenir de meilleurs résultats visuels.



Attention !

Le navigateur Firefox semble, pour l'instant, très mal supporter les dégradés radiaux lorsque les centres sont décalés.

Nous avons, dans notre exemple, utiliser un dégradé avec **quatre couleurs** positionnées respectivement à l'extrémité du dégradé, à 20 %, à 30 % puis à 50 %. Enfin, les dégradés radiaux peuvent également être **affectés à un contour** grâce à la propriété *strokeStyle*.

Utiliser une ombre portée

JAVASCRIPT

```
contexte2D.beginPath();
contexte2D.fillStyle = "rgb(255, 0, 0)";
contexte2D.shadowColor = "rgb(200, 200, 200)";
contexte2D.shadowOffsetX = 5;
contexte2D.shadowOffsetY = 5;
contexte2D.shadowBlur = 10;
contexte2D.fillRect(160, 160, 80, 80);
contexte2D.closePath();
```

Une ombre portée peut être placée sous une forme grâce à ces quatre propriétés :

- ▶ **propriété shadowColor** : couleur de l'ombre.
- ▶ **propriété shadowOffsetX** : décalage horizontal de l'ombre.
- ▶ **propriété shadowOffsetY** : décalage vertical de l'ombre.
- ▶ **propriété shadowBlur** : étendu du flou appliqué à l'ombre.

Seule la propriété *shadowColor* combinée avec un décalage sur l'ombre est obligatoire.

Afficher du texte

Et oui, en plus des formes, la balise <canvas> peut afficher du texte. Et comme pour les formes, ce dernier peut contenir un fond et un contour, comportant une couleur unie, un dégradé linéaire voire radial, ou encore posséder une ombre portée.

JAVASCRIPT

```
contexte2D.beginPath();
contexte2D.lineWidth = 2;
contexte2D.font = "bold italic 60px comic sans ms";
contexte2D.fillStyle = "rgb(255, 255, 0)";
contexte2D.strokeStyle = "rgb(255, 0, 255)";
contexte2D.fillText("coucou", 100, 200);
contexte2D.strokeText("coucou", 100, 200);
contexte2D.closePath();
```

La propriété *font* de <canvas> possède une **syntaxe identique à la propriété font en CSS**, et accepte donc toutes les valeurs existantes des propriétés CSS font-weight, font-style, font-variant, font-size et font-family.

L'ordre des instructions est strict : il faut débuter par la graisse et le style, puis intégrer le corps, et enfin la police avec son nom complet mais insensible à la casse. La police doit exister chez l'internaute (poste client), sinon c'est la police par défaut du navigateur qui s'affichera.

Deux méthodes sont utilisées pour afficher du texte :

- **méthode fillText** : permet d'afficher un texte avec un fond.
- **méthode strokeText** : permet d'afficher le contour d'un texte.

Ces deux méthodes utilisent des paramètres identiques :

- le texte à afficher ;
- la position x de la **ligne de base** du texte ;
- la position y de la **ligne de base** du texte.

Concernant l'utilisation des dégradés, la position que vous spécifiez pour le texte est la position de la ligne de base, donc le texte sera en réalité affiché au-dessus de la position spécifiée ; il faut s'en souvenir lorsque vous créez un dégradé recouvrant le texte.



Attention !

Rappelez-vous que la balise <canvas> permet de créer des images dynamiques, le texte affiché sera donc considéré comme une simple image par un moteur de recherche, et de ce fait, ne sera pas efficace pour améliorer le positionnement du site dans les pages de résultats des moteurs de recherche.

Importer une image

Après l'affichage du texte vient naturellement l'importation d'images externes. Celles-ci peuvent être utilisées dans un canevas comme une image seule, ou comme motif de remplissage dans une forme. Tous les types d'images acceptés par un navigateur Web peuvent être importés : JPG, PNG, GIF (fixe) et même SVG.

Affichage d'une image

JAVASCRIPT

```
// import d'une image
var importImage = new Image();
importImage.src = "image.jpg";
// fin de chargement de l'image
$(importImage).load(function(){
    contexte2D.drawImage(...);
});
```

L'importation d'une image externe passe, tout d'abord, par la création d'une variable de **type Image en JavaScript** qui permet d'indiquer le chemin de l'image à charger, et surtout d'y attacher un événement de fin de chargement. En effet, comme tout fichier externe, il est nécessaire d'attendre que le **chargement soit totalement terminé** avant d'accéder à son contenu. Pour cela, nous utilisons l'événement **load** du jQuery. Une fois le chargement terminé, c'est la **méthode drawImage** qui va afficher l'image dans le canevas. Cette méthode peut prendre plusieurs formes selon le type d'utilisation que vous souhaitez :

- affichage d'une image sans redimensionnement ;
 - affichage d'une image redimensionnée ;
 - création d'un masque pour n'afficher qu'une partie de l'image.
- Afficher une image sans redimensionnement

JAVASCRIPT

```
contexte2D.drawImage(  
    importImage,  
    40, 60  
)
```

Pour afficher une image externe sans redimensionnement, la **méthode drawImage** comprendra trois paramètres :

- la variable JavaScript de type Image ;
 - la position x de l'image dans le canevas ;
 - la position y de l'image dans le canevas.
- Afficher une image redimensionnée

JAVASCRIPT

```
contexte2D.drawImage(  
    importImage,  
    40, 60,  
    50, 50  
)
```

Pour redimensionner l'image chargée, la méthode drawImage va contenir **deux paramètres supplémentaires** :

- la largeur de l'image ;
- la hauteur de l'image.

Dans notre exemple, nous souhaitons créer une vignette de notre image, nous avons donc défini une largeur de 50 pixels et une hauteur de 50 pixels.

□ Création d'un masque

JAVASCRIPT

```
contexte2D.drawImage(  
    importImage,  
    40, 60,  
    50, 50,  
    175, 25,  
    50, 50  
)
```

Avec la création d'un masque, les paramètres de la **méthode drawImage se modifient totalement** :

- la variable JavaScript de type Image ;
- la position x du point initial du masque **dans l'image** ;
- la position y du point initial du masque **dans l'image** ;
- la largeur du masque **dans l'image** ;
- la hauteur du masque **dans l'image** ;
- la position x de l'image **dans le canevas** ;
- la position y de l'image **dans le canevas** ;
- la largeur de l'image **dans le canevas** ;
- la hauteur de l'image **dans le canevas**.

Seul le premier paramètre n'a pas changé.

Les paramètres deux à cinq permettent d'indiquer la position du point initial du masque et ses dimensions, mais en se conformant à des **coordonnées situées dans l'image** et non dans le canevas. De plus, **le masque doit recouvrir une partie existante de l'image**, si ce n'est pas le cas, certains navigateurs comme Chrome ou Safari, afficheront un canevas vide.

Les paramètres six à neuf permettent, quant à eux, de définir la position et les dimensions de l'image masquée dans le canevas. Il est préférable que les dimensions que vous spécifiez pour le masque dans l'image soient identiques aux dimensions de l'image affichée dans le canevas, pour éviter que le navigateur ne déforme l'image.

Image comme motif de remplissage

JAVASCRIPT

```
// import d'un motif
var importMotif = new Image();
importMotif.src = "motif.png";
// fin de chargement du motif
$(importMotif).load(function (){
var motif = contexte2D.createPattern(importMotif, "repeat");
contexte2D.beginPath();
contexte2D.fillStyle = motif;
contexte2D.fillRect(0, 0, 400, 400);
contexte2D.closePath();
});
```

La création d'un motif de remplissage ressemble à la création d'un dégradé. Il faut créer une variable JavaScript qui va stocker le motif en utilisant la **méthode createPattern** avec deux paramètres :

- la variable JavaScript du motif ;
- le type de répétition du motif.

Les valeurs possibles pour le type de répétition du motif sont identiques aux valeurs CSS pour la répétition d'une image de fond. Ces valeurs possibles sont :

- **valeur no-repeat** : pas de répétition du motif, il ne s'affichera qu'une seule fois.
- **valeur repeat** : le motif se répétera en mosaïque.
- **valeur repeat-x** : répétition uniquement horizontale.
- **valeur repeat-y** : répétition uniquement verticale.

Transformation d'une forme

Toutes nos formes n'ont jusqu'à présent subi aucune transformation mais vous souhaitez peut-être appliquer des rotations ou des déformations à celles-ci. En réalité, nous n'allons pas transformer une forme lors de sa création, mais plus exactement **transformer le contexte du dessin**, c'est-à-dire appliquer une rotation ou un dépla-

cément au canevas. Les transformations effectuées sur le canevas sont appliquées à tous les dessins jusqu'à sa réinitialisation.

Sauvegarde et restauration de l'état du canevas

JAVASCRIPT

```
// état1
contexte2D.fillRect(25, 175, 50, 50);
contexte2D.save();
// état2
contexte2D.fillStyle = "rgb(0, 0, 255)";
contexte2D.fillRect(75, 175, 50, 50);
contexte2D.save();
// état3
contexte2D.fillStyle = "rgb(0, 255, 255)";
contexte2D.fillRect(125, 175, 50, 50);
contexte2D.save();
// état4
contexte2D.fillStyle = "rgb(255, 0, 255)";
contexte2D.fillRect(175, 175, 50, 50);
// restauration de l'état3
contexte2D.restore();
contexte2D.fillRect(225, 175, 50, 50);
// restauration de l'état2
contexte2D.restore();
contexte2D.fillRect(275, 175, 50, 50);
// restauration de l'état1
contexte2D.restore();
contexte2D.fillRect(325, 175, 50, 50);
```

Dès que le canevas est transformé, **sa réinitialisation est nécessaire** lorsque la transformation n'est plus souhaitée. En effet, si vous effectuez la rotation d'une forme, vous ne voulez pas que toutes vos formes suivantes subissent la même rotation, d'où la nécessité de sauvegarder et de restaurer l'état du canevas.

Cette sauvegarde se fait avec la **méthode save**, et la restauration se fait avec la **méthode restore**. Les sauvegardes successives créent un **empilement d'états** et les restaurations ne font que revenir au dernier état sauvegardé. Les sauvegardes gardent en mémoire les transformations du canevas, mais aussi la couleur de remplissage et de contour, sur les formes et les textes. Pour comprendre l'empilement des états, regardons notre JavaScript de plus près :

- l'**état 1** du canevas correspond à l'état initial de tout canevas, le rectangle dessiné est donc rempli avec la couleur noire.
- la **méthode save** sauvegarde cet état du canevas.
- la couleur de remplissage est modifiée avec la couleur bleue, puis un rectangle est dessiné avec cette même couleur, et enfin cet état du canevas est lui aussi sauvegardé.
- à ce moment du code JavaScript, deux états sont sauvegardés, l'**état 1** avec le remplissage de couleur noire, et l'**état 2** avec la couleur de remplissage bleue.
- un nouveau rectangle est dessiné, et une nouvelle couleur de remplissage de couleur turquoise est sauvegardée, c'est l'**état 3**.
- le dernier état est l'**état 4** avec une couleur de remplissage fuchsia mais celui-ci n'est pas sauvegardé.

Il y a donc **trois états du canevas sauvegardés** à ce stade du code JavaScript : l'état 1 avec la couleur noire, l'état 2 avec la couleur bleue et l'état 3 avec la couleur turquoise.

- la **méthode restore** permet de **restaurer le dernier état sauvegardé** du canevas et de le **supprimer de la pile des états** ; donc, à ce moment du code, le dernier état sauvegardé est l'état 3 ; c'est pourquoi le rectangle dessiné suite à cette restauration reprend la couleur de remplissage de l'état 3 qui est la couleur turquoise.
- la suite du code reprend le même principe ; la deuxième utilisation de la méthode restore permet de revenir à l'état 2 d'où la couleur bleue du rectangle créé par la suite.
- la dernière utilisation de la méthode restore permet de revenir à l'état 1, le rectangle dessiné est donc rempli de noir.

Rotation du canevas

JAVASCRIPT

```
contexte2D.save();
contexte2D.rotate((10 * Math.PI) / 180);
contexte2D.fillRect(100, 100, 50, 150);
contexte2D.restore();
```

La **méthode rotate** permet d'effectuer la rotation d'une forme, et nécessite en paramètre, l'angle en radians de la rotation. Dans l'exemple ci-dessus :

- la **méthode save** est utilisée pour sauvegarder l'état du canevas lorsque celui-ci n'a pas subi de transformation.
- le canevas est transformé avec une rotation de 10° grâce à la **méthode rotate**.
- le rectangle est dessiné selon la rotation du canevas.
- la **méthode restore** permet de restaurer l'état initial du canevas, et ainsi de continuer à dessiner sans rotation.

Le point de rotation se trouve en haut à gauche de la forme.

JAVASCRIPT

```
contexte2D.rotate(degresEnRadians(10));
```

Il est toujours possible d'utiliser la fonction de conversion entre les radians et les degrés créée précédemment dans ce chapitre.

Déplacement du canevas

JAVASCRIPT

```
contexte2D.save();
contexte2D.translate(150, 100);
contexte2D.fillRect(0, 0, 50, 50);
contexte2D.restore();
```

La **méthode translate** permet de déplacer le point d'origine du canevas, placé par défaut en haut à gauche. Cette méthode utilise deux paramètres :

- la nouvelle position x de l'origine ;
- la nouvelle position y de l'origine.

Il est toujours important de revenir au point d'origine initial en sauvegardant et en restaurant l'état du canevas. Dans notre exemple, le point d'origine a été déplacé à la position 150 en x et 100 en y. De ce fait, le point initial de dessin du rectangle, se situant à 0 en x et 0 en y, correspond aux nouvelles coordonnées du canevas.

Mise à l'échelle du canevas

JAVASCRIPT

```
contexte2D.save();
contexte2D.scale(0.5, 1);
contexte2D.arc(400, 200, 80, 0, Math.PI * 2);
contexte2D.fill();
contexte2D.restore();
```

La **méthode scale** permet d'effectuer une mise à l'échelle du canevas et nécessite deux paramètres :

- ▶ l'échelle horizontale ;
- ▶ l'échelle verticale.

Ces paramètres sont des chiffres entiers ou des décimaux en **base 1**. La valeur 1, ou 100 %, signifie que le canevas ne subit aucune mise à l'échelle. Une mise à l'échelle du canevas influe sur la position et la taille des formes dessinées.

L'exemple présente une mise à l'échelle horizontale de 50 % et verticale de 100 % ; ainsi, le cercle s'est transformé en ovale. Pour centrer cet ovale, il faut en prendre en compte plusieurs conditions :

- ▶ si le canevas ne subit aucune mise à l'échelle, le centre se situe à 200 pixels en x et 200 pixels en y.
- ▶ si le canevas a subi une mise à l'échelle horizontale de 50 %, il faut que sa position x soit doublée, d'où la valeur de 400 pixels pour le point initial en x du dessin du cercle.
- ▶ Si le canevas n'a subi aucune mise à l'échelle verticale, le point initial en y du dessin du cercle reste à 200 pixels.

Matrice de transformation

JAVASCRIPT

```
contexte2D.save();
contexte2D.transform(
  2, 0.2,
  0.5, 2,
  -225, -180
);
contexte2D.fillRect(150, 150, 100, 100);
contexte2D.restore();
```

Une matrice de transformation sert à effectuer des **transformations géométriques** sur une forme. Deux méthodes agissent sur la matrice de transformation :

- **méthode transform** : la matrice de transformation s'additionne avec les autres transformations possibles telles que *rotate* ou *translate*.
- **méthode setTransform** : la matrice de transformation n'interagit pas avec les autres transformations activées au préalable.

Ces deux méthodes requièrent les mêmes paramètres :

- mise à l'échelle horizontale ;
- inclinaison horizontale ;
- inclinaison verticale ;
- mise à l'échelle verticale ;
- déplacement horizontal ;
- déplacement vertical.

Ces paramètres sont des chiffres entiers ou décimaux qui agissent comme multiplicateurs de la taille originelle de la forme. Dans notre exemple :

- la mise à l'échelle horizontale est de 2, donc la forme fait 200 % de sa largeur initiale.
- l'inclinaison horizontale est de 0,2, donc la forme subit une inclinaison de 30 pixels ($150 \times 0,2$).
- l'inclinaison verticale est de 0,5, donc une inclinaison de 75 pixels.
- la mise à l'échelle verticale est de 2, donc la forme fait 200 % de sa hauteur initiale.
- le déplacement horizontal de la forme est de -225 pixels par rapport à sa position horizontale initiale, qui était de 150 pixels lors de sa création.
- le déplacement vertical de la forme est de -180 pixels par rapport à sa position verticale initiale.

Réinitialisation de la matrice de transformation**JAVASCRIPT**

```
contexte2D.transform(  
    1, 0,  
    0, 1,  
    0, 0  
);
```

L'utilisation des paramètres ci-dessus avec la **méthode transform** sert à réinitialiser la matrice de transformation d'une forme. Cette dernière reprend son échelle initiale, sans aucune inclinaison, et sa position initiale.

Effacer des pixels

JAVASCRIPT

```
contexte2D.save();  
contexte2D.arc(200, 200, 50, 0, Math.PI * 2);  
contexte2D.fill();  
contexte2D.translate(200, 200);  
contexte2D.clearRect(-20, -20, 40, 40);  
contexte2D.restore();
```

Une méthode est disponible pour effacer les pixels du canevas dans une zone rectangulaire définie, c'est la **méthode clearRect**. Ses paramètres sont :

- la position x du point initial de la zone à effacer ;
- la position y du point initial de la zone à effacer ;
- la largeur de la zone à effacer ;
- la hauteur de la zone à effacer.



Principales API JavaScript

Découvrons les interfaces de programmation (en anglais API, pour *Application Programming Interface*) liées à l'univers du HTML5 mais qui, en réalité, ne sont pas du HTML5. Comme dans le chapitre précédent, nous allons continuer d'utiliser la **bibliothèque jQuery** pour le code JavaScript.

La géolocalisation

L'une des fonctionnalités les plus intéressantes liées au HTML5 est la possibilité de **géolocaliser l'internaute**. Celle-ci ouvre des perspectives intéressantes dans les domaines du **géomarketing** ou du simple guidage.

Les méthodes de géolocalisation

Le navigateur va utiliser **différentes méthodes** de géolocalisation qui dépendent de l'appareil utilisé et de ses possibilités.

Le GPS

Le moyen auquel l'on pense en premier pour la géolocalisation est évidemment le GPS. Néanmoins, ce dernier n'est **disponible que sur les appareils mobiles** – mobiles ou tablettes – et sera donc indisponible sur un ordinateur de bureau. Il fournit les résultats de positionnement les plus précis et un ensemble de paramètres, comme la vitesse ou l'angle par rapport au pôle magnétique nord, qui lui sont uniques.

□ Le réseau mobile

Le réseau mobile est utilisé lorsque le téléphone ne dispose pas du GPS ou lorsque vous vous trouvez à l'intérieur d'un bâtiment. Cette méthode de positionnement utilise les antennes relais et présente des résultats assez précis.

□ Le réseau WIFI

Le navigateur peut également utiliser le **réseau WIFI environnant** pour déterminer la position de l'internaute. Cette méthode de géolocalisation sera employée pour les appareils mobiles ainsi que pour les ordinateurs portables. Certes, les résultats de géolocalisation seront relativement précis mais ils ne fourniront que les données de base, à savoir la longitude, la latitude et la précision du positionnement.

□ L'adresse IP

La dernière méthode de géolocalisation est l'adresse IP, qui sera principalement utilisée par les ordinateurs de bureau. Le navigateur interroge une base de données contenant les correspondances entre les IP existantes et leur localisation. L'adresse IP provenant généralement d'un **nud de raccordement au réseau**, la localisation va aboutir, en réalité, à la position de ce nœud de raccordement, ce qui engendre des résultats très imprécis.

Testez la localisation de votre IP

Pour tester votre positionnement avec la méthode de l'adresse IP, consultez la page <http://whatismyipaddress.com/>

Découverte de l'interface de programmation

L'objet JavaScript *navigator* a été étendu pour posséder désormais une propriété *geolocation*. Cette propriété dispose de trois méthodes :

- **getCurrentPosition** : fournit une géolocalisation unique.
- **watchPosition** : permet de récupérer la position de manière continue.
- **clearWatch** : permet de stopper la géolocalisation continue.

HTML

```
<!DOCTYPE HTML>
<html lang="fr">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-
scale=1.0, user-scalable=no" />
<title>Géolocalisation</title>
<!-- JQuery -->
<script type="text/javascript" src="jquery.js"></script>
<!-- JavaScript -->
<script type="text/javascript"></script>
</head>
<body>
<article><article>
</body>
</html>
```

Notre page HTML importe la bibliothèque jQuery, puis utilise une **balise <script>** qui nous permettra d'écrire le code JavaScript nécessaire à l'utilisation de la géolocalisation, et enfin, emploie une **balise <article>** pour afficher les informations liées à la géolocalisation.

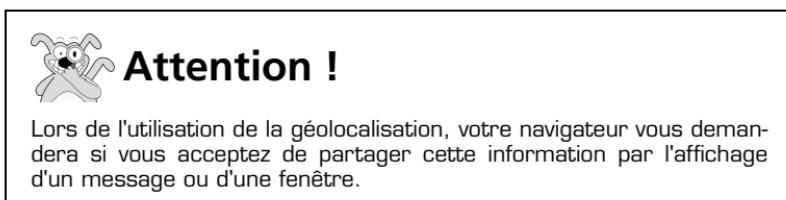
❑ Compatibilité du navigateur

JAVASCRIPT

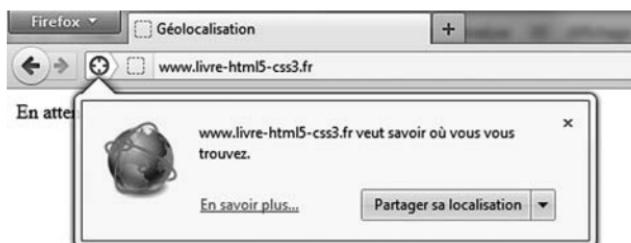
```
<script type="text/javascript">
// attente du chargement complet de la page HTML
$(document).ready(init);
// initialisation
function init(e){
    // test de la compatibilité
    if(navigator.geolocation){
        $("article").html("Votre navigateur gère la
géolocalisation.");
    }
    else{
        $("article").html("Votre navigateur ne gère pas la
géolocalisation.");
    }
}
</script>
```

Avant toute chose, pour utiliser la géolocalisation, il faut que le navigateur client gère cette possibilité, d'où la nécessité de **tester la compatibilité** du navigateur.

Dans notre code, après avoir attendu le chargement complet de la page HTML nécessaire pour avoir accès à son contenu, nous appelons une fonction d'initialisation nommée *init*. Dans cette fonction, nous testons la présence de la propriété *geolocation* de l'objet JavaScript *navigator*. Si le test n'est pas concluant, un message d'erreur s'affichera dans la balise *<article>*. En revanche, si le navigateur client gère la géolocalisation, nous allons pouvoir localiser l'internaute à l'aide de la méthode **getCurrentPosition**.



Exemple sur Internet Explorer



Exemple sur Firefox

Géolocalisation unique**JAVASCRIPT**

```
if(navigator.geolocation){  
  
    navigator.geolocation.getCurrentPosition(localisationReussie);  
    $("article").html("En attente de la localisation...");  
}
```

La méthode **getCurrentPosition** nécessite un paramètre obligatoire et deux optionnels.

Commençons par le premier paramètre qui est l'appel d'une fonction se déclenchant lorsque la **localisation a réussi**.

JAVASCRIPT

```
// la localisation a réussi  
function localisationReussie(informationsPosition){  
    $("article").html("Nous vous avons localisé.");  
}
```

Cette fonction **contient les informations de positionnement à travers** un paramètre de type *Position* nommé dans notre exemple **informationsPosition**. Ce paramètre possède deux propriétés :

- propriété *coords* : récupère toutes les coordonnées de la position.
- propriété *timestamp* : calcule le tampon temporel, c'est-à-dire le moment où le positionnement a réussi.

**À savoir**

Un tampon temporel est le nombre de secondes écoulées depuis le 1^{er} janvier 1970 à minuit. Cette date sert de référence à l'informatique car elle est considérée comme étant la date de naissance du système Unix.

□ Récupération des coordonnées du positionnement

JAVASCRIPT

```
function localisationReussie(informationsPosition){  
    // propriété coords de l'objet informationsPosition  
    $("article").html(  
        "Latitude : " + informationsPosition.coords.latitude + "° <br />" +  
        "Longitude : " + informationsPosition.coords.longitude + "° <br />" +  
        "Précision du positionnement : " +  
        informationsPosition.coords.accuracy + " mètres <br />"  
    );  
  
    // propriété timestamp de l'objet informationsPosition  
    $("article").append(  
        "Positionnement effectué le : " + informationsPosition.timestamp  
    );  
}
```

La propriété *coords* possède elle-même des propriétés :

- propriété *latitude* : récupère la latitude en degrés décimaux.
- propriété *longitude* : récupère la longitude en degrés décimaux.
- propriété *accuracy* : détermine la précision du positionnement en mètres.



Exemple sur Internet Explorer

On s'aperçoit ici de la précision du positionnement par l'adresse IP !



Exemple sur une tablette galaxy tab avec GPS actif

□ Le gestionnaire d'erreur

JAVASCRIPT

```
navigator.geolocation.getCurrentPosition(localisationReussie,  
localisationEchouee);  
// la localisation a échoué  
function localisationEchouee(informationsErreur){  
    // messages d'erreur  
    var messagesErreur = {  
        "1" : "Permission de localisation refusée.",  
        "2" : "Position indéterminée.",  
        "3" : "Permission de localisation expirée."  
    };  
  
    // affichage du message d'erreur  
    $("article").html(  
        messagesErreur[informationsErreur.code]  
    );  
}
```

Nous avons jusqu'ici supposé que tout se passait dans les meilleures conditions, mais ce n'est pas toujours le cas... **et si vous étiez dans un tunnel**, le GPS ne serait plus actif.

La gestion des erreurs est donc nécessaire pour indiquer à l'internaute qu'une erreur de géolocalisation est survenue. Il faut savoir que le gestionnaire d'erreur est le deuxième paramètre possible de la méthode **getCurrentPosition**. Il est optionnel mais fortement recommandé !

La fonction **localisationEchouee** reçoit elle-même un paramètre de type *GeoPositionError* qui indique le type d'erreur survenue.

La propriété *code* du paramètre **informationsErreur** récupère le code d'erreur sous la forme d'un chiffre de 1 à 3 :

- *code d'erreur 1* (correspondant à `PERMISSION_DENIED`) : indique que l'internaute n'a pas autorisé sa localisation.
- *code d'erreur 2* (correspondant à `POSITION_UNAVAILABLE`) : indique que la localisation n'a pu être déterminée.
- *code d'erreur 3* (correspondant à `TIMEOUT`) : indique que le délai imparti à la localisation a expiré.

Pour passer ce message en français, nous avons créé un objet dont les propriétés sont les codes d'erreur numériques, et dont les valeurs sont la transcription des erreurs en français.

□ Options de configuration de la géolocalisation

JAVASCRIPT

```
var optionsGeolocalisation = {  
    enableHighAccuracy : true,  
    timeout : 30000,  
    maximumAge : 60000};  
navigator.geolocation.getCurrentPosition(localisationReussie,  
localisationEchouee, optionsGeolocalisation);
```

Le troisième et dernier paramètre possible pour la méthode **getCurrentPosition** concerne les options de la géolocalisation. C'est un objet JavaScript avec trois propriétés :

- propriété *enableHighAccuracy* : permet de demander une précision accrue de positionnement, ce qui aura généralement pour effet de mettre en route le GPS de l'appareil. Ce paramètre attend **une valeur booléenne**.
- propriété *timeout* : permet de déterminer le délai maximum de positionnement. Au-delà de ce délai, la géolocalisation échoue. Ce paramètre est exprimé en **millisecondes**.
- propriété *maximumAge* : permet de spécifier la durée durant laquelle une position est considérée comme valide et est gardée en mémoire par le navigateur. Ce paramètre est exprimé en **millisecondes**.



À savoir

Une valeur booléenne est binaire, elle n'accepte que les valeurs *true* (vrai) ou *false* (faux).

Dans notre exemple, nous avons choisi une durée maximale de positionnement de 30 secondes et **une validité de la dernière position récupérée de 1 minute**.

Géolocalisation continue**JAVASCRIPT**

```
var identifiantPositionnement =
navigator.geolocation.watchPosition(localisationReussie,
localisationEchouee, optionsGeolocalisation);
$("article").html(
"Latitude : " + informationsPosition.coords.latitude + "° <br />" +
"Longitude : " + informationsPosition.coords.longitude + "° <br />" +
+
"Précision du positionnement : " +
informationsPosition.coords.accuracy + " mètres <br />" +
"Altitude : " + informationsPosition.coords.altitude + " mètres <br />" +
+
"Précision de l'altitude : " +
informationsPosition.coords.altitudeAccuracy + " mètres <br />" +
"Angle par rapport au pôle magnétique nord : " +
informationsPosition.coords.heading + "° <br />" +
"Vitesse : " + informationsPosition.coords.speed + " mètres par
seconde <br />"
);
```

Si vous souhaitez « pister » votre internaute, il faut utiliser la méthode **watchPosition** en lieu et place de la méthode **getCurrentPosition**. Cette méthode retourne également un **identifiant numérique unique de positionnement** qui sert, par la suite, à stopper la localisation.

Avec la géolocalisation continue, de nouvelles propriétés de coordonnées s'activent :

- propriété *altitude* : permet de connaître l'altitude en mètres.
- propriété *altitudeAccuracy* : permet de connaître la précision de l'altitude en mètres.
- propriété *heading* : permet de récupérer l'angle, de 0 à 360 degrés, par rapport au pôle magnétique nord.
- propriété *speed* : permet de connaître la vitesse de déplacement en mètres par seconde.

Si ces propriétés affichent la valeur *null*, c'est-à-dire vide, cela signifie que, soit le GPS de votre appareil ne peut récupérer la propriété, soit celle-ci est indisponible.

□ Stopper une géolocalisation continue

HTML

```
<input type="button" id="bouton" value="Stopper la géolocalisation"
/>
```

JAVASCRIPT

```
$("#bouton").click(stopperGeolocalisation);
// stopper la localisation
function stopperGeolocalisation(e){

    navigator.geolocation.clearWatch(identifiantPositionnement);
}
```

Ajoutez un bouton dans le HTML, puis en JavaScript, dans la fonction **init** ajoutez un événement clic sur le bouton.

Pour finaliser notre exemple, nous utilisons la méthode **clearWatch** qui permet de stopper une géolocalisation continue. Cette méthode prend en paramètre l'identifiant de positionnement que nous avons défini grâce à la méthode **watchPosition**, et que nous avons nommé *identifiantPositionnement*.

Le stockage des données

Un des inconvénients du protocole HTTP est qu'il ne garde pas d'informations en mémoire entre les pages HTML. Pourtant, cette fonctionnalité est primordiale dans beaucoup de sites. Par exemple, si vous avez acheté vos produits préférés dans votre supermarché en ligne, vous ne voudriez pas perdre tous vos achats précédents en changeant de page.

Avant le HTML5

Pour combler cette lacune, il existe principalement **deux systèmes de stockage de données** : les cookies et les sessions PHP.

Les **cookies** sont de petits fichiers texte stockés dans votre ordinateur, qui gardent en mémoire des informations pour un temps défini par le créateur du cookie.

Les **sessions PHP**, à l'inverse des cookies, stockent des informations tant que votre navigateur est démarré, puis les « oublie » quand il s'éteint.

Découverte de l'interface de programmation

Grâce au HTML5, ces deux techniques sont vouées à disparaître, car désormais vous avez la possibilité d'utiliser l'**API Web Storage**, qui se divise en deux nouvelles propriétés de l'objet JavaScript `window` :

- propriété `sessionStorage` : stockage de données par session, équivalent aux sessions PHP, qui stockent des informations tant que le navigateur est en marche.
- propriété `localStorage` : stockage local de données, équivalent aux cookies, qui stockent des informations même après la fermeture du navigateur.

Ces deux propriétés possèdent **cinq mêmes méthodes** :

- méthode `setItem` : création d'une entrée.
- méthode `getItem` : récupération de la valeur d'une entrée.
- méthode `removeItem` : suppression d'une entrée.
- méthode `key` : récupération du nom d'une clé par sa position dans la liste des entrées stockées, la position 0 correspond à la première entrée.
- méthode `clear` : suppression de toutes les entrées.

L'**unique propriété** disponible est la propriété `length`, qui permet de calculer le nombre d'entrées stockées en mémoire.

Caractéristiques de Web Storage

L'API Web Storage permet de créer un **nombre illimité d'entrées** par nom de domaine. Pour des raisons évidentes de sécurité, celles-ci ne sont **accessibles que par le nom de domaine « créateur »**.

Les valeurs stockées sont considérées comme des données de type `String`, c'est-à-dire des chaînes de caractères. Si vous stockez un chiffre et que vous souhaitez l'utiliser pour des opérations arithmétiques, il faudra passer par la méthode JavaScript `parselInt`, qui permet de convertir une donnée de type `String` en une donnée de

type *Number*. De même, si vous souhaitez stocker des informations plus complexes comme des objets ou des tableaux, il faudra les sérialiser, c'est-à-dire les convertir en chaîne de caractères, à l'aide de la fonction JavaScript **JSON.stringify**, et les désérialiser grâce à **JSON.parse**. Le poids total des informations stockées va jusqu'à **5 Mo**.

Le stockage de données par session

HTML

```
sessionStorage01.html
<!DOCTYPE HTML>
<html lang="fr">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-
scale=1.0, user-scalable=no" />
<title>Tout pour bien débuter avec HTML5 et CSS3 - Principales API
JavaScript - Stockage de données</title>
<!-- JQuery -->
<script type="text/javascript" src="jquery.js"></script>
<!-- JavaScript -->
<script type="text/javascript"></script>
</head>
<body>
<article>
<form action="sessionStorage02.html" id="formulaire">
Prénom : <input type="text" id="champPrenom" required />
Nom : <input type="text" id="champNom" required />
<input type="submit" value="valider" />
</form>
</article> </body>
</html>

sessionStorage02.html
<!DOCTYPE HTML>
<html lang="fr">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-
scale=1.0, user-scalable=no" />
<title>Tout pour bien débuter avec HTML5 et CSS3 - Principales API
JavaScript - Stockage de données</title>
<!-- JQuery -->
<script type="text/javascript" src="jquery.js"></script>
<!-- JavaScript -->
```

```
<script type="text/javascript"></script>
</head>
<body>
<article></article>
</body>
</html>
```

Dans notre exemple, nous souhaitons récupérer les noms et prénoms de nos internautes par l'intermédiaire d'un formulaire. La première page nommée **sessionStorage01.html** contient ce formulaire, la seconde page nommée **sessionStorage02.html** va afficher les données stockées. Nous allons utiliser le stockage de session, les informations saisies seront donc effacées après la fermeture du navigateur.

□ Compatibilité du navigateur

JAVASCRIPT

```
sessionStorage01.html
<script type="text/javascript">
// attente du chargement complet de la page HTML
$(document).ready(init);
// initialisation
function init(e){
    // test de la compatibilité
    if(window.sessionStorage){
        $("article").html("Votre navigateur gère le stockage de
données.");
    }
    else{
        $("article").html("Votre navigateur ne gère pas le stockage
de données.");
    }
}
</script>
```

Un test de compatibilité du navigateur est toujours nécessaire pour déterminer s'il supporte l'API Web Storage. Dans le cas contraire, une erreur s'affiche dans la balise <article>.



Attention !

Sur les navigateurs Firefox et Internet Explorer, vous devez obligatoirement tester le stockage de données par session à l'aide d'un serveur Web local. Avec les autres principaux navigateurs comme Opera, Chrome ou Safari, vous pouvez le tester sans l'aide d'aucun serveur Web local.

- Serveurs Web gratuits sur Windows :
EasyPHP : <http://www.easyphp.org/fr/>
- WampServer : <http://www.wampserver.com/>
- xampp : <http://www.apachefriends.org/fr/xampp.html>
- Serveur Web gratuit sur Mac OS X :
MAMP : <http://www.mamp.info/en/>

Création d'une entrée

JAVASCRIPT

```
sessionStorage01.html
if(window.sessionStorage){
    $("#formulaire").submit(stockageDonnees);
}
```

Nous lisons à l'événement *submit*, c'est-à-dire l'envoi du formulaire, une fonction nommée **stockageDonnees**.

JAVASCRIPT

```
sessionStorage01.html
// stockage des données
function stockageDonnees(e){
    window.sessionStorage.setItem("stockagePrenom",
        $("#champPrenom").val());
    window.sessionStorage.setItem("stockageNom",
        $("#champNom").val());
}
```

Nous utilisons la méthode **setItem** de l'objet *sessionStorage* pour créer une entrée. Cette méthode nécessite deux paramètres, communément appelés clé/valeur :

- ▶ la **clé** est le nom de l'entrée ;
- ▶ la **valeur** est le contenu de l'entrée.

Dans notre exemple, nous avons créé **deux entrées** ayant pour clé *stockagePrenom* et *stockageNom*, et pour valeur le contenu du champ de saisie s'y rapportant.

□ Récupération de la valeur d'une entrée

JAVASCRIPT

```
sessionStorage02.html
<script type="text/javascript">
// attente du chargement complet de la page HTML
$(document).ready(init);
// initialisation
function init(e){
    // affichage des données
    if(
        window.sessionStorage.getItem("stockagePrenom")
        && window.sessionStorage.getItem("stockageNom")
    ){
        $("article").html(
            "Votre prénom est : " +
            window.sessionStorage.getItem("stockagePrenom") + ".<br />" +
            "Votre nom est : " +
            window.sessionStorage.getItem("stockageNom") + "."
        );
    }
    else{
        $("article").html("Nous ne connaissons ni votre prénom, ni votre
nom.");
    }
}
</script>
```

La méthode **getItem**, recevant en paramètre le nom de la clé ciblée, permet de récupérer la valeur de cette même clé. Dans notre exemple, nous avons également utilisé la méthode **getItem** pour tester l'existence des clés *stockagePrenom* et *stockageNom*.

□ Suppression d'une ou plusieurs entrées

HTML

```
sessionStorage02.html
<input type="button" id="bouton" value="Supprimer les données
stockées" />
```

```
javascript
sessionStorage02.html
$("#bouton").click(supprimerDonnees);
// supprimer les données
function supprimerDonnees(e){
    window.sessionStorage.clear();
    window.location.reload();
}
```

Ajoutez un bouton dans le HTML, puis en JavaScript, dans la fonction **init**, ajoutez un événement clic sur le bouton. Celui-ci est lié à la fonction **supprimerDonnees** qui utilise la méthode **clear** pour supprimer toutes les données stockées.

Si vous ne souhaitez **supprimer qu'une seule entrée**, il faut utiliser la méthode **removeItem** avec en paramètre le nom de la clé à effacer. Par exemple, si vous souhaitez supprimer uniquement la clé *stockagePrenom* : `window.sessionStorage.removeItem ("stockagePrenom")`.

Enfin, nous demandons à la page de se recharger pour qu'elle en prenne en compte la suppression des données.

Le stockage local de données

Le stockage local de données utilise les **mêmes méthodes** que le stockage de données par session. La différence est que l'on va utiliser l'objet *localStorage* en remplacement de l'objet *sessionStorage*. Rappelons que le stockage local de données conserve les informations après fermeture du navigateur.



Attention !

Sur le navigateur Internet Explorer, vous devez obligatoirement tester le stockage local de données à l'aide d'un serveur Web local. A contrario, sur les autres navigateurs, aucune installation de serveur Web local n'est nécessaire.

Pour notre exemple, imaginons un formulaire à remplir si les noms et prénoms n'ont pas été stockés dans une saisie précédente, ou un remplissage automatique des champs si ces informations ont déjà été stockées localement.

Le code HTML est identique à l'exemple utilisé pour le stockage de données par session, à la différence que l'**attribut action du formulaire** pointe sur la même page que celle contenant le formulaire.

JAVASCRIPT

```
<script type="text/javascript">
// attente du chargement complet de la page HTML
$(document).ready(init);
// initialisation
function init(e){
    // test de la compatibilité
    if(
        window.localStorage
        && !window.localStorage.getItem("stockagePrenom")
        && !window.localStorage.getItem("stockageNom")){
            $("#formulaire").submit(stockageDonnees);
    }
    else if(
        window.localStorage
        && window.localStorage.getItem("stockagePrenom")
        && window.localStorage.getItem("stockageNom")){
            remplissageChamps();
    }
    else{
        $("article").html("Votre navigateur ne gère pas le stockage de
données.");
    }
}
// stockage des données
function stockageDonnees(e){
window.localStorage.setItem("stockagePrenom",
$("#champPrenom").val());
window.localStorage.setItem("stockageNom", $("#champNom").val());
}
//remplissage des champs du formulaire
function remplissageChamps(){
$("#champPrenom").val(window.localStorage.getItem("stockagePrenom"));
$("#champNom").val(window.localStorage.getItem("stockageNom"));
}
</script>
```

La fonction **init** gère trois possibilités :

- si le navigateur est compatible avec l'API Web Storage mais que les clés *stockagePrenom* et *stockageNom* n'existent pas : la saisie est stockée grâce dans fonction **stockageDonnees** à l'aide de la méthode **setItem**.
- si le navigateur est compatible et que les deux clés existent : la fonction **remplissageChamps** récupère la valeur des clés avec la méthode **getItem** et les affiche dans les champs du formulaire.
- si le navigateur est incompatible : un message d'erreur s'affiche.

Pour tester le stockage local de données, saisissez vos prénom et nom, cliquez sur le bouton « valider », puis fermez votre navigateur, et enfin, rouvrez-le pour constater que les champs du formulaire ont été pré-remplis.



Positionnement & accessibilité

Avant tout, il est peut-être utile de revenir sur la définition de ces deux termes :

- **Positionnement** (en anglais, *Search engine optimization* : SEO) : regroupe les diverses techniques permettant à un site d'apparaître le mieux positionné possible, dans les pages de résultats des moteurs de recherche type Google, Yahoo ou Bing. Cette mise en œuvre se fait dès la conception du site et durant toute la phase d'écriture du code et des contenus. Attention à ne pas confondre positionnement et référencement, ce dernier étant l'action de notifier (par l'envoi d'un formulaire numérique en général) l'existence d'un site auprès des moteurs de recherche ou d'annuaires Internet généralistes ou spécialisés.
- **Accessibilité** : ensemble des techniques mises en œuvre, comme pour le positionnement, lors de la conception et de l'écriture du code, et permettant la consultation d'un site par un handicapé au moyen d'un terminal de lecture adapté, tel un clavier braille ou un lecteur audio pour les non-voyants.

Même si les deux termes désignent des domaines bien différents, on a souvent tendance à les associer car d'une part, ils partagent certaines techniques et d'autre part, ce qui est bon pour l'un est bon pour l'autre. On peut, par exemple, évoquer l'ajout de textes alternatifs aux images : techniques d'amélioration de l'accessibilité consistant à ajouter un court texte descriptif de l'image. Les interfaces de lecture seront alors à même de « lire » ce texte, permettant ainsi à la personne handicapée d'avoir accès à l'information.

Mais ces textes alternatifs sont aussi sources d'information pour les moteurs de recherche, car grâce à leur raison d'être, ils renferment quantité de mots-clés (du moins si l'intégrateur connaît son métier), qui sont des indicateurs descriptifs du site.

En résumé, comme nous le verrons dans ce chapitre, les nouveautés apportées par l'HTML5 bénéficient surtout au positionnement dans les moteurs de recherche, et indirectement, participent à une meilleure accessibilité.

Métadonnées, Microdatas, Rich Snippets...

Il s'agit de termes qui visent tous la même chose : tenter de classer, établir une nomenclature, analyser les contenus des pages, en faisant ressortir les éléments *sensibles*, *attrayants*, *marquants* tels que le nombre d'étoiles d'une recette de cuisine, une photo de la recette, les coordonnées d'un contact, l'éditeur d'un livre, etc. En anglais, on les nomme *Rich Snippets*.

Les Microdatas sont l'outil adapté à leurs traitements, même si on en est qu'au balbutiement. Ils désignent le nouveau système de balisage d'HTML5 pour marquer les contenus « sensibles » permettant aux applications Web mais également aux moteurs de recherche de mieux appréhender la catégorie sémantique des contenus qu'ils parcouruent.

Que du bonheur donc pour Google, Bing ou Yahoo, puisque grâce à ces Microdatas, ils peuvent enrichir les annonces de résultats de leurs pages de recherche, en insérant dans ces annonces des données sensibles ! Indirectement, cela leur permet sans doute de mieux positionner les pages en interprétant plus facilement les contenus, mais il y a surtout l'effet optique : les annonces intégrant des *Rich Snippets* sont graphiquement plus visibles que les autres (par exemple, les étoiles d'avis attirent l'œil).

La meilleure recette de la pâte à crêpes - Les recettes les mieux ...

> ... > Recette pâte à crêpe à la bière

★★★★★ 180 votes

Retrouvez les la meilleure recette de la pâte à crêpes notées ...

31	Pâte à crêpes à la	20	attente : 3 heures,
bière	minutes	cuisson	
4	Crêpe Normande	20	attente : 3 heures,
votes	minutes	cuisson	

**Attention !**

Ne vous emballez pas :

- d'une part, les moteurs de recherche ne garantissent pas l'inclusion de ces données dans leurs résultats (mais ils le font pour l'instant).
- d'autre part, ils affirment ne pas en tenir compte pour positionner les sites !

Il convient donc de suivre l'affaire car elle a tout de même son importance, et sera sans doute un élément important dans le positionnement des sites dans le futur. Ainsi, les firmes éditrices de ces moteurs de recherche se sont associées pour fournir à la communauté un vocabulaire adapté au Microdatas ([shema.org](#)), vocabulaire qu'elles espèrent bien voir devenir un standard. D'ailleurs, le passé risque de leur donner raison, car cela a déjà été réalisé, et avec succès, pour les sitemaps : [sitemap.org](#). En attendant ce jour, il est de toute façon évident que des sites qui auront fait des efforts pour structurer leurs contenus mais aussi pour les marquer (par les microdatas en utilisant le vocabulaire préconisé dans [shema.org](#)) verront leur investissement récompensé.

API History

HTML5 met à disposition des développeurs une nouvelle API : l'API History.

Mais que désigne-t-elle exactement ? C'est une abréviation pour désigner les termes : *Applications Programming Interface*, ce qui

pourrait se traduire par interface de programmation, entendez par là une trousse à outils de base, permettant de programmer une application Web.

Un exemple ?

L'API Google Map qui permet à un développeur d'utiliser dans son site ou son application le programme permettant la géolocalisation.

HTML5 History API

HTML5 History API available

Last event fired: popstate

To test the History API, click through the urls below. Note that none of these urls point to *real* pages. JavaScript will intercept these clicks, load data and the browser address bar will *appear* to change - but this is the History API in action!

Use the back and forward buttons in your browser to navigate the history.

- [first](#)
- [second](#)
- [third](#)
- [fourth](#)

Que permet cette API History ?

La modification dynamique de l'URL affichée dans la barre d'adresse du visiteur, sans pour cela changer le contenu de la page ou être obligé de recharger la page. Cette application est très pratique pour afficher simplement et rapidement les contenus demandés par l'internaute, suite à une requête issue d'un champ de recherche. Par ailleurs, elle inscrit également cette adresse dans la mémoire du navigateur, autorisant ainsi une navigation basée sur l'historique, précédent et suivant.

(Voir démo : <http://html5demos.com/history>).

Ceci représente un réel progrès car pour les pages de sites utilisant de l'Ajax, ce genre de navigation est impossible, la page n'étant pas rechargée. Rien n'est parfait ! Idem pour les favoris, qui deviennent gérables grâce à cette API. Tout cela entraîne donc, indirectement, une meilleure prise en compte des pages par les moteurs de recherche, puisque ceux-ci sont depuis longtemps intéressés par les libellés des URL.

L'influence de la structuration sémantique du code par les nouvelles balises HTML5

On n'a pas attendu l'HTML5 pour avoir des balises ayant un rôle sémantique, c'est-à-dire des balises qui, par leur désignation, indiquent aux moteurs de recherche le contenu qu'elles mettent en page. Les balises <H> en sont un parfait exemple puisqu'elles sont conçues pour recevoir des éléments de titres et sous-titres. Les titres comportant souvent des mots-clés, c'est donc tout naturellement que les moteurs de recherche accordent beaucoup d'importance à ce type de balises.

Avec son lot de nouvelles balises, l'HTML5 apporte de nombreuses possibilités d'amélioration du référencement naturel, en permettant, voire en obligeant, une meilleure structuration des données dans la page, ce qui conduit également à une hiérarchisation effective des informations. Ces nouvelles balises sont :

- ▶ <header> qui se traduit par « en-tête ». Son nom est donc suffisamment explicite : c'est la partie haute de votre page de site, comportant généralement un logo, la date du jour, une navigation persistante, un bref texte de présentation, etc., qui va définir le header de votre page/section.



Attention !

À ne pas confondre avec la balise <HEAD>, qui, elle, date des débuts de l'HTML, et qui se place avant la balise <body>. Elle est destinée à fournir des informations au navigateur comme les liens de chargement des fichiers associés (CSS, JS), le titre de la page et diverses déclarations. Donc, tout contenu inséré dans cette balise ne sera pas visible dans la page (erreur classique de débutant !).

- ▶ <section> définit une partie de page. Les autres balises HTML5 sont donc en général incluses dans cette balise, puisque <section> intervient pour définir les grandes zones d'un article.

- ▶ <article> définit une contribution, un post ou un commentaire ou tout autre contenu important de la page.
- ▶ <aside> définit un contenu en rapport avec la section, mais qui n'apparaîtra pas dans la page. Il ne fait pas réellement partie du contenu ; il n'est là que pour apporter des précisions ou des éclaircissements sur le contenu lui-même. Un exemple : si le contenu-texte est une recette, utilisant ce qu'on appelle en cuisine un « appareil », c'est-à-dire une préparation culinaire comportant au moins deux ingrédients préalablement mélangés, vous pourrez mettre en <aside> cette définition d'un appareil.
- ▶ <hgroup> est une balise chargée de regrouper les balises de titre, les balises <h1> à <h6>. Elle s'avère idéale par exemple pour un plan de site.
- ▶ <nav> est chargée de contenir les listes de liens.
- ▶ <figure> est utile pour insérer une image avec sa légende.
- ▶ <figcaption> doit être obligatoirement insérée dans la balise <figure> et permet de placer une légende pour l'illustration présente dans la balise <figure>.
- ▶ <footer> est semblable à <header> mais définit cette fois-ci le bas ou pied de page.

Type de champ « search »

Dans les champs de recherche ou dans les formulaires, HTML5 permet d'apporter un nouveau type d'input : « search ». Cela équivaut à une sorte de spécialisation de ce champ.

Par ailleurs, en plaçant à la suite, l'attribut « placeholder », vous déclenchez l'inscription du contenu de cet attribut dans le champ de recherche. Ce contenu pourra jouer le rôle d'indicateur ou d'aide à la saisie du champ. Notez qu'il disparaîtra dès que l'internaute aura cliqué sur le champ en vue de le renseigner.

Bien que ce genre d'opérations était déjà possible dans l'ère pré-HTML5 en utilisant du JavaScript, désormais c'est direct, donc plus simple et plus accessible aux moteurs de recherche, qui n'apprécient pas toujours le JavaScript. Il en résulte que ces contenus pré-indicateurs sont tout à fait lisibles par les moteurs de

recherche ce qui ouvre de nouveaux espaces pour apporter des éléments susceptibles d'améliorer le positionnement.

```
<form>
<input type="search" placeholder="Tapez votre annonce" />
    <input type="submit" />
</form>
```

Tapez votre annonce

Envoyer

Du nouveau dans les liens avec l'attribut <rel>

Le Web est souvent comparé, et même appelé la Toile, en raison de la notion d'hyperliens qui a prévalu à sa création, et donc qui est présente depuis son origine. Tous les sites ne fonctionnent que grâce à ces liens qui permettent de relier plusieurs ressources ensemble, et ainsi de naviguer d'un point à un autre. De par leur importance, ces liens sont donc aussi pour les moteurs de recherche une source inestimable pour mieux appréhender un site et tenter d'en mesurer la popularité et l'audience. Seulement jusqu'à présent, faute de réels paramètres pour les décrire, Google s'intéressait plus à la valeur des cibles qu'à la nature du lien. La preuve en est avec les backlinks, ces liens de « partenaires ou d'amis » que vous pouvez déposer sur un site, et qui sont à la base de la notion de « page Rank » ou indice de popularité d'une page.

Quelques exemples de description de liens existent tout de même depuis longtemps, comme celle utilisée pour les déclarations de feuilles de style :

```
<link rel="stylesheet" href="feuilles-style.css" type="text/css">
```

Ou bien encore, dans la création de liens en «nofollow» :

```
<a href="http://www.site-partenaire.com" rel="nofollow">Visitez le
site de notre partenaire</a>
```

Avec HTML5, il est désormais possible d'aller plus loin dans la description de la nature de ces liens, grâce à de nouvelles valeurs appliquées à l'attribut « rel » des balises `<a>`, `<link>` et `<area>`, les 3 balises de liens. Parmi les nombreuses propositions de valeurs, quatre sont particulièrement précieuses en termes de SEO :

- "rel=search" qui indique le lien de la ressource permettant d'effectuer les recherches demandées. Cela représente donc également un moyen pour les moteurs de recherche d'aller chercher des informations, et non pas de se contenter de pages de résultats de recherche qu'ils ignorent le plus souvent, considérant que celles-ci s'apparentent à des pages dupliquées.
`Recherche...`
- "rel=tag" qui affecte le mot-clé au document courant en signalant par là même la relation entre différentes ressources du document principal. (Attention : il n'y a aucun rapport avec une déclaration de mots-clés des métadonnées). Le mot-clé désigné ici sera d'ordre sémantique, et fait donc référence à un espace de noms (*namespace*) [qui est une zone définie et virtuelle, renfermant l'ensemble des termes appartenant à un tronc commun].
`monproduit`
- "rel=prev" et "rel=next"
Pour les sites ayant de nombreuses pages (annuaires, forums), et afin que les moteurs de recherche puissent établir l'ordre de pagination (pas toujours évident et lisible pour eux quand seuls certains paramètres changent dans les URL, et notamment derrière le « ? »), de nouvelles valeurs viennent désormais enrichir un attribut déjà connu : le "rel" des href.

Ainsi, au lieu d'écrire :

```
| <a href="produits.php?page=12">Page précédente</a>
| <a href="produits.php?page=13">Page suivante</a>
```

On écrira :

```
| <a href="produits.php?page=12" rel="prev"> Page précédente</a>
| <a href="produits.php?page=13" rel="next"> Page suivante</a>
```

Grâce à cela, Google ou Bing comprendront que les pages font partie d'un même ensemble de produits. Il s'agit sans doute là d'un détail dans l'ensemble des tâches à effectuer pour apporter un bon positionnement mais c'est aussi en multipliant ce genre de détails que l'on arrive à bien positionner un site.



Les nouveaux sélecteurs CSS

Une page HTML est une arborescence où les balises aussi appelées éléments forment une structure ayant des liens de parenté.

Les sélecteurs CSS permettent de **cibler des éléments précis** situés dans cette arborescence par l'utilisation d'une panoplie de conditions symbolisées par des caractères différents, par exemple le dièse cible un identifiant. Malgré l'utilisation du nom générique de « sélecteurs CSS », il faut distinguer 3 groupes :

- ▶ **les sélecteurs** : ils permettent de cibler des éléments par des classes, des attributs ou leur emplacement dans la page HTML.
- ▶ **les pseudo-classes** : ils permettent de cibler un élément qui n'est pas dans l'arborescence de la page HTML.
- ▶ **les pseudo-éléments** : ils permettent de cibler une partie d'un élément présent dans l'arbre de la page HTML.

Depuis la version 3 des CSS, le W3C recommande, pour différencier **pseudo-classes et pseudo-éléments**, d'utiliser le caractère « deux-points » pour les pseudo-classes, et deux caractères « deux-points » pour les pseudo-éléments.

Les sélecteurs CSS3

Les sélecteurs d'attributs

Les sélecteurs d'attributs permettent de **cibler une balise par la valeur d'un de ses attributs**. La norme CSS3 introduit **trois**

nouveaux sélecteurs d'attributs, qui se différencient par la position de la chaîne de caractères recherchée dans la valeur de l'attribut :

- **E[A[^]="C"]** : le caractère « caret » indique le ciblage d'un élément E dont la valeur de l'attribut A débute par la chaîne de caractères C.
- **E[A\$="C"]** : le caractère « dollar » indique le ciblage d'un élément E dont la valeur de l'attribut A se termine par la chaîne de caractères C.
- **E[A*="C"]** : le caractère « astérisque » indique le ciblage d'un élément E dont la valeur de l'attribut A contient la chaîne de caractères C.

□ Sélecteur d'attribut unique

HTML

```
<article>
  <p>
    <a href="document.zip">lien zip interne</a>
  </p>
</article>
```

CSS

```
a[href$=".zip"]){
  color:rgb(255, 0, 255);
}
```

L'exemple ci-dessus permet de cibler les balises **<a>** dont l'attribut **href** se termine par la chaîne de caractères « **.zip** ».

□ Sélecteur d'attributs multiples

HTML

```
<article>
  <p>
    <a href="document.doc">lien doc interne</a>
    <br />
    <a href="http://www.site.fr/document.doc">lien doc externe</a>
  </p>
</article>
```

CSS

```
a[href$=".doc"]{  
background:rgb(0, 255, 255);  
}  
a[href^="http"][href$=".doc"]{  
background:rgb(255, 255, 0);  
}
```

Le ciblage d'une balise par son attribut peut être encore plus précis lorsque vous utilisez des **attributs multiples**. Dans notre exemple, nous avons différencié les fichiers DOC qui sont sur votre serveur et ceux qui sont sur un serveur externe, par l'utilisation de deux sélecteurs d'attributs :

- l'attribut *href* doit débuter par « http » ;
- l'attribut *href* doit finir par « .doc ».

Le sélecteur d'adjacence indirecte

HTML

```
<h1>Titre</h1>  
<p>paragraphe 1</p>  
<h2>Sous-titre</h2>  
<p>paragraphe 2</p>  
<blockquote>  
<p>paragraphe 3</p>  
</blockquote>
```

CSS

```
h1 ~ p{  
font-size:1.5em;  
font-style:italic;  
}
```

Le sélecteur d'adjacence indirecte permet de **sélectionner les éléments précédés, immédiatement ou pas, par un élément frère**, c'est-à-dire possédant le même parent. La syntaxe est la suivante :

E ~ F : le caractère tilde permet de cibler les éléments F précédés par l'élément E.

Dans notre exemple, le sélecteur d'adjacence indirecte permet de cibler les deux paragraphes précédés de la balise **<h1>**, y compris

celui qui est précédé de la balise **<h2>** puis qu'ils possèdent tous deux la balise **<h1>** comme frère. Quant au paragraphe 3, il n'est pas ciblé car son parent direct est la balise **<blockquote>** et non la balise **<h1>**.

Les pseudo-classes CSS3

L'énième enfant

Parmi un ensemble d'éléments possédant le même parent, comme les éléments d'une liste ou les lignes d'un tableau, il est fastidieux en CSS2 de cibler un élément particulier par sa position. Pour remédier à cette difficulté, les CSS3 introduisent le **ciblage d'un énième enfant d'une balise parente**.

Quatre nouvelles pseudo-classes ont ainsi été créées :

- **P E:nth-child(position)** : cible l'énième enfant E de l'élément parent P, en débutant le comptage de la position par le premier enfant, et en descendant dans la structure HTML.
- **P E:nth-last-child(position)** : cible l'énième enfant E de l'élément parent P, en débutant le comptage de la position par le dernier enfant, et en remontant dans la structure HTML.
- **P E:nth-of-type(position)** : cible l'énième occurrence du type de l'élément E présent dans l'élément parent P, en débutant le comptage de la position par le premier enfant de type E, et en descendant dans la structure HTML.
- **P E:nth-last-of-type(position)** : cible l'énième occurrence du type de l'élément E présent dans l'élément parent P, en débutant le comptage de la position par le dernier enfant de type E, et en remontant dans la structure HTML.

La **position de l'élément ciblé** se spécifie avec un paramètre qui peut être de trois types différents :

- **un nombre entier positif** : représente une position unique dont le comptage a débuté à 1.
- **un multiplicateur** : représente des multiples pouvant contenir une opération arithmétique.
- **un mot-clé** : représente les positions paires ou impaires.

Un entier positif en paramètre**HTML**

```
<article>
  <h1>Titre</h1>
  <p>paragraphe1</p>
  <p>paragraphe2</p>
  <p>paragraphe3</p>
  <p>paragraphe4</p>
  <ul>
    <li>élément1</li>
    <li>élément2</li>
    <li>élément3</li>
    <li>élément4</li>
    <li>élément5</li>
  </ul>
</article>
```

CSS

```
article p:nth-child(3){
  font-style:italic;
}
article p:nth-last-child(2){
  color:rgb(255, 0, 0);
}
article li:nth-of-type(3){
  background:rgb(255, 255, 0);
}
article li:nth-last-of-type(1){
  text-decoration:underline;
}
```

Si le nombre spécifié en paramètre de ces pseudo-classes est positif, il représentera la **position exacte de l'élément par rapport à sa balise parente**. Par exemple :

- **article p:nth-child(3)** : cible le troisième enfant de la balise parente ici la balise `<article>` si celui-ci est un paragraphe ; dans notre exemple, ce sélecteur cible le « `paragraphe2` ».
- **article p:nth-last-child(2)** : cible l'avant-dernier enfant de la balise parente si celui-ci est un paragraphe ; dans notre exemple, ce sélecteur cible le « `paragraphe4` ».

- **article li:nth-of-type(3)** : cible le troisième enfant de la balise parente de type `` ; dans notre exemple, ce sélecteur cible l'« élément3 ».
 - **article li:nth-last-of-type(1)** : cible le dernier enfant de la balise parente de type `` ; dans notre exemple, ce sélecteur cible l'« élément5 ».
- Un multiplicateur en paramètre

HML

```
<article>
  <ul>
    <li>élément1</li>
    <li>élément2</li>
    <li>élément3</li>
    <li>élément4</li>
    <li>élément5</li>
    <li>élément6</li>
    <li>élément7</li>
    <li>élément8</li>
  </ul>
</article>
```

CSS

```
article li:nth-child(4n){
  list-style-type:none;
}
article li:nth-child(3n + 2){
  border:1px solid rgb(0, 0, 0);
}
article li:nth-last-child(-n + 3){
  font-weight:bold;
}
```

Le nombre en paramètre peut être un **multiplicateur symbolisé par le caractère « n »**, débutant à la valeur 0 et s'incrémentant de 1. Utilisé avec un chiffre le précédent, ce dernier cible des multiples.



Attention !

Le multiplicateur « n » ne doit pas être séparé de son multiple par un espace.

Dans notre exemple, le paramètre « 4n » cible :

- **la position 4** : résultat du calcul $(4 * 1)$.
- **la position 8** : résultant du calcul $(4 * 2)$.

Le paramètre peut également être une **opération arithmétique**.

Dans notre exemple, l'opération arithmétique « 3n + 2 » cible :

- **la position 2** : résultant du calcul $(3 * 0) + 2$.
- **la position 5** : résultant du calcul $(3 * 1) + 2$.
- **la position 8** : résultant du calcul $(3 * 2) + 2$.

Enfin, il est possible d'utiliser un **multiplicateur négatif**, qui a pour effet d'inverser le sens de sélection des éléments. Celui-ci fait remonter la sélection dans la structure HTML, alors qu'un multiplicateur positif la fait descendre. Dans notre exemple, l'opération arithmétique « -n + 3 » combinée à la pseudo-classe « nth-last-child » cible :

- **la position 3** : résultant du calcul $(0 + 3)$;
- **la position 2** : résultant du calcul $(-1 + 3)$;
- **la position 1** : résultant du calcul $(-2 + 3)$.

La pseudo-classe « nth-last-child » fait démarrer le comptage des éléments à partir de la fin de la structure HTML, et l'opération arithmétique fournit la position de l'élément à cibler.

Un mot-clé en paramètre

HTML

```
<table>
<tr>
<td>texte1</td>
<td>texte2</td>
<td>texte3</td>
</tr>
```

```
<tr>
<td>texte4</td>
<td>texte5</td>
<td>texte6</td>
</tr>
<tr>
<td>texte7</td>
<td>texte8</td>
<td>texte9</td>
</tr>
<tr>
<td>texte10</td>
<td>texte11</td>
<td>texte12</td>
</tr>
</table>
```

CSS

```
article tr:nth-child(even){
background:rgb(200, 200, 200);
}
```

Deux mots-clés existent pour cibler facilement certains énièmes enfants :

- ▶ **odd** : cible les positions impaires ; c'est l'équivalent de l'opération arithmétique $2n + 1$.
- ▶ **even** : cible les positions paires ; c'est l'équivalent de l'opération arithmétique $2n$.

Les premiers et derniers enfants

HTML

```
<article>
<h1>Titre</h1>
<p>paragraphe1</p>
<kbd>paragraphe2</kbd>
<p>paragraphe3</p>
<blockquote>paragraphe4</blockquote>
<kbd>paragraphe5</kbd>
</article>
```

CSS

```
article:last-child{
border-bottom:5px solid rgb(0, 0, 0);
```

```
}

article kbd:first-of-type{
padding:15px;
background:rgb(200, 200, 200);
}
article p:last-of-type{
border-bottom:1px solid rgb(0, 0, 0);
}
```

En CSS2 existe la pseudo-classe « first-child » qui permet de cibler le premier enfant d'un élément parent.

En CSS3, trois nouvelles pseudo-classes ont été créées pour **cibler spécifiquement les premiers et derniers enfants** :

- **E:last-child** : cible le dernier enfant de l'élément E.
- **E:first-of-type** : cible la première occurrence du type de l'élément E contenue dans une balise parente.
- **E:last-of-type** : cible la dernière occurrence du type de l'élément E contenue dans une balise parente.

Par exemple :

- **article:last-child** : cible le dernier enfant de la balise <article> ; dans notre exemple, ce sélecteur cible le « paragraphe5 ».
- **article kbd:first-of-type** : cible la première occurrence de la balise <kbd> contenue dans la balise parente <article> ; dans notre exemple, ce sélecteur cible le « paragraphe2 ».
- **article p:last-of-type** : cible la dernière occurrence de la balise <p> contenue dans la balise parente <article> ; dans notre exemple, ce sélecteur cible le « paragraphe3 ».

Les enfants uniques

HTML

```
<article>
<h1>Article</h1>
<p>
<a href="#">commentaire1</a>
</p>
<p>
<a href="#">commentaire2</a>
par <kbd>utilisateur1</kbd>
```

```
</p>
<p>
<a href="#">commentaire3</a>
par <kbd>utilisateur2</kbd> et <kbd>utilisateur3</kbd>
</p>
</article>
```

CSS

```
article a:only-child{
color:rgb(255, 0, 0);
}
article kbd:only-of-type{
background:rgb(255, 0, 0);
}
```

Deux pseudo-classes CSS3 permettent de **cibler les enfants uniques** :

- **E:only-child** : cible l'élément lorsqu'il est l'unique enfant de l'élément E.
- **E:only-of-type** : cible l'élément lorsqu'il est l'unique occurrence du type de l'élément E.

Par exemple :

- **article a:only-child** : cible la balise `<a>` unique enfant de sa balise parente ; dans notre exemple, ce sélecteur cible le « commentaire1 ».
- **article kbd:only-of-type** : cible la balise `<kbd>` unique occurrence de ce type dans sa balise parente ; dans notre exemple, ce sélecteur cible l'« utilisateur1 ».

La cible d'un lien

HTML

```
<article>
<div id="bloc1">bloc1</div>
<div id="bloc2">bloc2</div>
<div id="bloc3">bloc3</div>
<div id="bloc4">bloc4</div>
<ul>
<li><a href="#bloc1">afficher le bloc1</a></li>
<li><a href="#bloc2">afficher le bloc2</a></li>
```

```
<li><a href="#bloc3">afficher le bloc3</a></li>
<li><a href="#bloc4">afficher le bloc4</a></li>
<ul>
</article>
```

CSS

```
#bloc1, #bloc2, #bloc3, #bloc4{
width:100px;
height:100px;
background:rgb(200, 200, 200);
position:absolute;
display:none;
}
article ul{
list-style-type:none;
position: absolute;
left:100px;
}
#bloc1{
display:block;
}
#bloc1:target, #bloc2:target, #bloc3:target, #bloc4:target{
display:block;
}
#bloc1:target ~ div, #bloc2:target ~ div, #bloc3:target ~ div,
#bloc4:target ~ div{
display:none;
}
```

La pseudo-classe **target** sélectionne un élément qui est la cible d'un lien par l'intermédiaire d'une ancre. Une ancre HTML est un lien interne à la page, reconnaissable par un dièse suivi d'un identifiant. Cette pseudo-classe fait obligatoirement intervenir deux éléments :

- l'élément à cibler qui doit posséder un identifiant ;
- le lien contenant une ancre vers l'identifiant de l'élément à cibler.

Sa syntaxe est la suivante :

E:target : permet de cibler l'élément E qui est lui-même la cible d'une ancre HTML.

Cette pseudo-classe permet une **réelle interactivité**, proche de certains comportements auparavant uniquement accessibles via du JavaScript, comme la création de diaporama ou de menu à onglets. Dans notre exemple, nous avons créé un menu à onglet. Pour cela, nous avons besoin d'élément à cibler possédant chacun un identifiant. C'est le cas pour nos éléments `<div>` nommés « bloc1 », « bloc2 », etc. Ces blocs sont superposés, et seul le « bloc1 » est visible au lancement de la page HTML.

Le menu est composé d'une liste de liens, qui contiennent chacun une ancre ciblant l'un des blocs par son identifiant. Lorsque l'internaute clique sur l'un des liens :

- l'identifiant contenu dans l'ancre est ciblé par la pseudo-classe **target** ;
- la pseudo-classe **target** fait afficher le bloc ciblé et cache tous les autres blocs.

La racine du document

CSS

```
html:root{  
background:rgb(200, 200, 200);  
}
```

La pseudo-classe **root** permet de cibler la racine de la page HTML, c'est-à-dire le premier élément du document. Autrement dit, cette pseudo-classe ne permet de cibler que la balise `<html>`. La syntaxe de **root** est la suivante :

E:root : cible la racine de l'élément E qui ne peut être que la balise `<html>`.

Les éléments vides

HTML

```
<table border="1">  
<tr>  
<th>en-tête 1</th>  
<th>en-tête 2</th>  
<th></th>  
<th><kbd></kbd></th>  
</tr>  
<tr>
```

```
<td>cellule 1</td>
<td>cellule 2</td>
<td>cellule 3</td>
<td>cellule 4</td>
</tr>
</table>
```

CSS

```
th:empty{
background:url(motif.gif);
}
```

La sélection d'un **élément vide ne contenant ni d'autres éléments ni de contenu textuel** se fait grâce à la pseudo-classe **empty**. Dans notre exemple, seul l'en-tête vide de tout contenu possède un motif de fond. L'en-tête de la quatrième colonne semble également vide, mais la balise **<kbd>** est néanmoins considérée comme un contenu. La syntaxe de la pseudo-classe **empty** est la suivante :

E:empty : permet de cibler l'élément E ne possédant aucun contenu.

La pseudo-classe de négation

HTML

```
<article>
<h1>titre</h1>
<p>paragraphe 1</p>
<p>paragraphe 2</p>
<h2>sous-titre</h2>
<p>paragraphe 3</p>
</article>
```

CSS

```
article *:not(p){
color:rgb(255, 0, 0);
}
```

La pseudo-classe de négation permet de **sélectionner des éléments à partir d'un sélecteur négatif**.

Dans notre exemple, la pseudo-classe permet de sélectionner toutes les balises enfants de **<article>** qui ne sont pas des balises **<p>**. La syntaxe est donc la suivante :

E:not(S) : permet de cibler l'élément E qui ne fait pas partie du sélecteur S.

Les pseudo-classes CSS3 de formulaires

Un ensemble de pseudo-classes a été créé en CSS3 pour améliorer l'aspect visuel des formulaires.

Les champs actifs, inactifs et cochés

HTML

```
<form>
<p>
Cases à cocher :
<br />
<input type="checkbox" /><label>Choix 1</label>
<input type="checkbox" /><label>Choix 2</label>
<input type="checkbox" disabled /><label>Choix 3</label>
</p>
<p>
<p>
<input type="submit" value="Valider" />
</p>
</form>
```

CSS

```
input[type="checkbox"]:enabled + label{
color:rgb(0, 0, 0);
}
input[type="checkbox"]:disabled + label{
opacity:0.2;
}
input[type="checkbox"]:checked + label{
color:rgb(0, 255, 0);
}
```

Les pseudo-classes **enabled**, **disabled** et **checked** permettent de cibler respectivement les champs actifs, inactifs et cochés d'un formulaire. Dans notre exemple, la balise **<label>** se modifie selon l'état de la case à cocher :

- **actif** : le texte est noir ;

- **inactif** : le texte est à 20 % d'opacité ;
- **coché** : le texte est vert.

Les champs obligatoires, valides et invalides

HTML

```
<form>
<p>
<label>Mail :</label>
<br />
<input type="email" required />
</p>
<p>
<input type="submit" value="Valider" />
</p>
</form>
```

CSS

```
input[type="email"]:required{
border:1px solid rgb(255, 0, 0);
}
input[type="email"]:valid{
border:1px solid rgb(0, 0, 0);
color:rgb(0, 0, 0);
}
input[type="email"]:invalid{
color:rgb(255, 0, 0);
}
```

Pour mettre en exergue les champs obligatoires d'un formulaire, on utilise la pseudo-classe **required** qui permet de cibler ce type de champs. Il est également possible de cibler les champs dont le contenu est valide ou invalide grâce aux pseudo-classes **valid** et **invalid**.

Dans notre exemple, le champ obligatoire de type email :

- est pourvu d'une bordure rouge pour montrer que le champ est obligatoire ;
- la bordure et le texte saisi dans le champ deviennent noirs, lorsque la saisie est valide ;
- le texte saisi est rouge tant que la saisie est invalide.

Les pseudo-éléments CSS3

Concernant les pseudo-éléments, seules **une nouvelle syntaxe et une nouveauté sont apparues avec les CSS3.**

En CSS2, les pseudo-éléments first-line, first-letter, before, after étaient précédés d'un unique caractère « deux-points ». **Désormais, la syntaxe officielle nécessite deux caractères « deux-points ».**

Précisons tout de même que l'ancienne syntaxe reste **rétro-compatibile**.

Le pseudo-élément de sélection

HTML

```
<article>
  <p>
    Sélectionnez ce texte !
  </p>
</article>
```

CSS

```
article p::selection {
background:rgb(255, 0, 0);
}
article p::-moz-selection {
background:rgb(255, 0, 0);
}
```

Le pseudo-élément **selection** est destiné à cibler **la partie sélectionnée d'un texte**. De ce fait, seules les propriétés de couleur de fond et de couleur de texte sont utilisables avec lui. La syntaxe est la suivante :

E::selection : permet de cibler la partie sélectionnée de l'élément E.

Il est à noter qu'à l'heure où nous écrivons ces lignes, le navigateur Firefox nécessite le **préfixe -moz-** pour l'utilisation de la pseudo-classe **selection**.



CSS3 : enrichissements graphiques

CSS3 regorge de nouveautés en tout genre, qui vont de l'enrichissement graphique simple (bordure arrondie, ombre, transparence) à des choses plus complexes telles que les animations, la gestion du timing, l'emploi de polices embarquées, etc. Malheureusement, comme cela a déjà été dit précédemment, toutes ces avancées ne sont encore prises en compte que partiellement par nos navigateurs ; néanmoins, tous les navigateurs récents en supportent un grand nombre. Bien sûr, et nous en avons fâcheusement l'habitude, il est un cas spécial qui s'appelle Internet Explorer, et qui ne daigne commencer à intégrer un tant soit peu les CSS3 qu'à partir de la version 9. Dommage, car il reste de nombreuses versions 7 et 8 dans les postes des entreprises. Seul le temps peut intervenir.

Pour savoir exactement qui supporte quoi, ces sites vous diront tout, que ce soient pour les CSS3, l'HTML5 ou certaines API : www.caniuse.com ou www.quirksmode.org

Et si en complément, vous souhaitez connaître les dispositions de votre navigateur usuel pour ces technologies, c'est ici : <http://css3test.com>

Commençons par découvrir les principales propriétés graphiques et ce qu'elles apportent. Seulement, pour l'instant, il faut encore du temps pour que ces dernières soient définitivement reconnues et interprétées correctement par tous les navigateurs. De fait, apporter des enrichissements graphiques requiert de taper un peu plus de code qu'avec les CSS2 dans la mesure où ces nouvelles

propriétés sont au départ des expérimentations établies par chaque famille de navigateurs (en fonction des directives W3C tout de même). Du coup, ces navigateurs préconisent de faire précéder les déclarations de propriétés d'un préfixe (dénommé « préfixes propriétaires » ou « préfixes vendeurs » !) propre à chaque navigateur ou plutôt à chaque moteur de rendu utilisé par ces navigateurs. Le tableau ci-dessous vous présente les principaux moteurs utilisés par les navigateurs, leurs correspondances et les préfixes à utiliser.

MOTEUR	NAVIGATEURS	PRÉFIXE
Trident	IE, Maxthon	-ms
Webkit	Safari, Chrome, Nav Androïd, iPhone	-webkit
Gecko	Firefox	-moz-
Presto	Opera, Dreamweaver	-o-

Ainsi, pour donner des bords arrondis à un cadre, fonctionnant sous les navigateurs Firefox, Opera, Chrome et IE9, voici ce que nous devrions écrire :

```
.coin_rond {
    -moz-border-radius: 8px;
    -webkit-border-radius: 8px;
    border-radius: 8px;
}
```

Cependant, rassurez-vous :

- avec le temps, plus ça va aller, moins vous aurez besoin de rajouter ces préfixes. D'ailleurs, déjà, pour border-radius, plus aucun navigateur (FF v11, Chrome v18, Opera v11, IE v9, Safari v4) n'a besoin des préfixes propriétaires.
- il existe un petit sauveur, sous forme d'un fichier JavaScript, qu'il suffit d'appeler en début de page, et qui va se charger de rajouter les préfixes pour vous. Vous trouverez cette perle à l'adresse suivante : <http://imsky.github.com/cssFx/>

Voir l'exemple dans le fichier : border-radius03.html

Les présentations et préludes étant faits, il est temps de découvrir ces propriétés.

Les Bordures de boîtes (**Border**)

Border-radius

Fini les boîtes aux coins droits, vive les coins arrondis. Vous me direz que vous n'avez pas attendu les CSS3 pour voir ce type de boîtes. En effet, sauf qu'avant, il fallait des lignes et des lignes de codes CSS2, des images pour représenter chaque coin, etc. Désormais, une propriété CSS3 et c'est terminé ; en réalité trois lignes, si vous rajoutez les préfixes, mais cette propriété n'a plus besoin de préfixes, tous les navigateurs la reconnaissent.

```
border-radius: 80px;
```

Avec les préfixes, on obtient :

```
-moz-border-radius: 80px;  
-webkit-border-radius: 80px;  
border-radius: 80px;
```

Voir l'exemple dans le fichier : border-radius01.html



```
border-radius: 80px;
```

(Petit rappel : si les 4 valeurs des coins (haut, droit, bas, gauche) sont égales, il suffit d'indiquer une seule des valeurs).

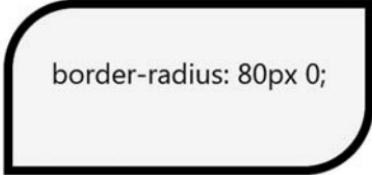
```
border-radius: 80px
```

Donc équivalent à :

```
border-radius: 80px 80px 80px 80px;
```

En variant les valeurs, on peut obtenir toutes sortes de possibilités.

Voir l'exemple dans le fichier : border-radius02.html



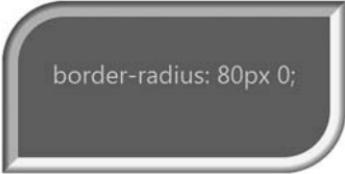
border-radius: 80px 0;

Border-colors

Toutefois, certaines propriétés ne peuvent encore être reconnues que par quelques navigateurs ; c'est le cas pour la propriété « `border-color` » qui ne fonctionne que sous Firefox. Ceci est fort dommage car elle permet de multiplier les teintes dans les bordures de boîtes, autorisant les dégradés par exemple.

Voir l'exemple dans le fichier : border-radius04.html

```
-moz-border-top-colors: #e00 #c30 #c50 #c60 #c70 #c80 #c90 #ca0  
#cb0 #cc0;
```



border-radius: 80px 0;

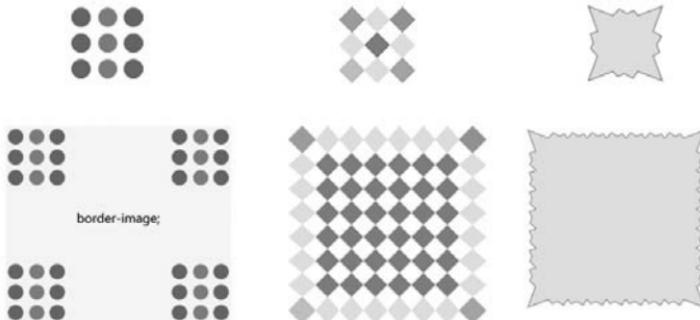
Border-images

Cette propriété promet de nombreuses possibilités quand elle sera acceptée par tous les navigateurs. Actuellement, seuls Firefox et Chrome l'ont au catalogue.

Imaginez : l'insertion d'images au sein des bordures, véritable ouverture vers de nombreux enrichissements.

Voir l'exemple dans le fichier : border-radius05.html.

Voici trois exemples (en haut l'image insérée dans la bordure et en-dessous, le résultat monté).



Box-shadow

En revanche, cette propriété est réalisable avec tous les navigateurs (à partir d'IE9, et Firefox, Chrome, Opera...).

Voir l'exemple dans le fichier : border-radius06.html.

border-shadow;

Box-shadow : largeur / décalage / flou / distance / couleur

Les Backgrounds

Quelques nouveautés intéressantes en CSS3, notamment dans le positionnement de ces fonds de boîtes, mais également dans le fait qu'ils peuvent désormais être multiples. Cela promet de beaux délires.

Commençons, par le premier, légèrement anecdotique je le reconnaît, surtout qu'il ne fonctionne pour l'instant que sous Chrome. Mais, ce qu'il laisse envisager est intéressant.

Backgrounds-clip

Permet de définir les limites des backgrounds, suivant trois variantes :

- « content » : reste dans les limites du contenu qui peut être réduit par un padding (figure ci-dessous) :

```
-webkit-border-image: url(img/puces.png) 200%;  
-webkit-background-image: url(img/bckgrd/bckgrd1.png);  
-webkit-background-clip: content;
```



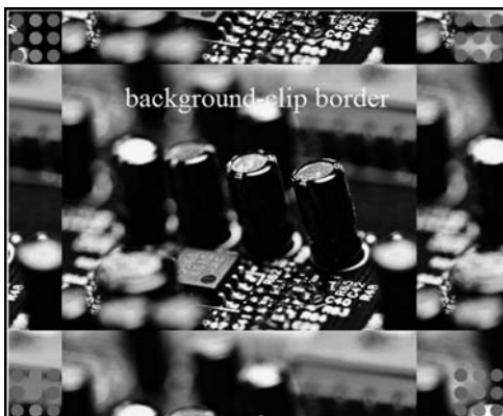
- « padding » : reste dans les limites du contenu sans tenir compte du padding (figure ci-après) :

```
-webkit-border-image: url(img/puces.png) 200%;  
-webkit-background-image: url(img/bckgrd/bckgrd1.png);  
-webkit-background-clip: padding;
```



- » « border » : le fond s'étend jusque sous les bordures (figure ci-dessous) :

```
-webkit-border-image: url(img/puces.png) 200%;  
-webkit-background-image: url(img/bckgrd/bckgrd1.png);  
-webkit-background-clip: border;
```



Voir ces 3 exemples dans le fichier : [background-clip.html](#)

Background-Size

Avec la CSS3, il est désormais possible de définir au pixel près (ou en pourcentage de l'image, avec comme point d'origine, la position définie par la valeur attribuée à « background-origin », autre nouveauté CSS3) la taille de vos images de fond, que ce soit en largeur ou en hauteur.

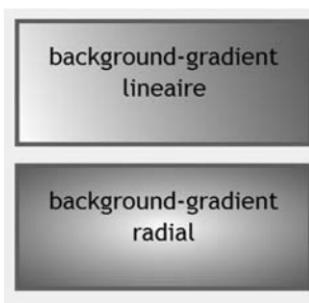
- `background-size: width height (cover/contain);`
- `cover` : l'image de fond couvre toute la boîte, quitte à être coupée en cas de surface de fond trop petite.
- `contain` : l'image de fond est contenue dans la surface définie mais sans déformation.

Background Dégradés

Grâce à ces nouvelles propriétés, j'ai malheureusement peur que tous ceux qui se seront penchés un peu trop sur le tonneau des CSS3, et qui à force seront tombés dedans, n'usent et n'abusent de ces dégradés désormais facilement définissables en fond d'image. Il faut reconnaître que cela va aussi bien simplifier la vie et laisser reposer un peu Photoshop.

Deux sortes de dégradés sont disponibles : linéaires ou radiaux.

Voir l'exemple dans le fichier : `background-grad.html`



`Background : linear-gradient (sens dégradé X et/ou sens y, couleur1 , ..., couleurN)`

Background Multiples

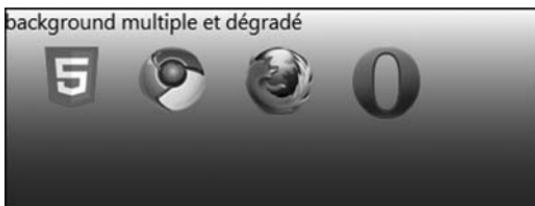
Pouvoir désormais, grâce aux CSS3, positionner plusieurs backgrounds, l'un sur l'autre, comme les calques d'un logiciel graphique comme Photoshop par exemple, va permettre de faciliter la vie des intégrateurs et des webdesigners pour les en-têtes de pages par exemple, concernant notamment des logos, devant apparaître sur des fonds différents.

C'est désormais très facile et surtout plus besoin d'aller créer une image sous Photoshop ou tenter de mettre des images les unes à côté des autres avec des listes horizontales par exemple. Il suffit de déclarer les divers background puis de les positionner. Vous pouvez même rajouter un petit dégradé pour le fun, si vous le désirez ou plutôt si votre client y tient énormément.

Voici un exemple de code avec préfixe propriétaire pour Firefox :

```
background:url(img/bckgrd/html5.png) 30px 30px no-repeat,  
url(img/bckgrd/chrome.png) 100px 30px no-repeat,  
url(img/bckgrd/ff.png) 180px 30px no-repeat,  
url(img/bckgrd/opera.png) 260px 30px no-repeat,  
-moz-linear-gradient(top center, white, blue);.
```

Et son résultat ci-dessous :



Text-Shadow

Cette propriété est une propriété finalisée, qui n'a donc pas besoin de préfixe. Par contre, elle n'est pas reconnue par IE, même IE9. Sa déclaration est simple :

```
text-shadow: décalage x décalage y largeur zone de flou couleur;
```

Pour proposer un dégradé, il suffit de déclarer les diverses couches de couleurs et leurs décalages.

Voir l'exemple dans le fichier : text-shadow.html

```
text-shadow:0 0 4px white, 0 -5px 4px yellow,  
2px -10px 6px pink,  
-2px -15px 11px cyan,  
2px -25px 18px magenta;
```



Opacité

Gardons le meilleur pour la fin, avec les réglages d'opacités de toute zone [texte comme dans l'exemple ci-dessous, images...], propriété permettant de réaliser des effets très tendances en webdesign actuels. Réaliser cela en CSS2 était possible mais au prix de nombreuses astuces et quelques lignes de code.

```
opacity: valeur comprise entre 0 (transparent) et 1 (opaque)  
Exemple : opacity: 0.2;
```



Transparence

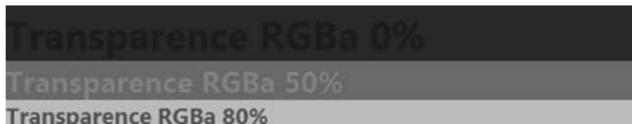
Cette notion ne doit pas être confondue avec la précédente. Si la transparence était déjà possible en CSS2, notamment avec les formats d'images PNG (à partir de la version 6 pour IE), il n'en était rien par contre pour les zones d'aplats de couleurs. Pour ce faire, les CSS3 ajoutent un paramètre supplémentaire à la définition de teintes RGB (ou RVB) qui devient RGBa.

Concernant le background-color :

```
R (rouge) = 0 ;  
G (vert) = 0 ;  
B (bleu) = 255 ;  
A (transparence) = 50 %.
```

```
.transp2 {  
    background-color: rgba(0, 0, 255, 0.5);  
    color: rgba(255,255,255,0.2);  
}
```

Voir l'exemple dans le fichier : transparence.html





Typographie et mise en page

Typographie

S'il y a bien une chose que l'on attendait avec impatience chez les concepteurs Web, c'est bien la possibilité de pouvoir intégrer directement les polices de son choix, dans les interfaces de sites.

Avant et jusqu'à présent, pour l'affichage de polices, il y avait deux choix

- travailler avec des polices vectorielles, ce qui bien sûr est l'idéal car c'est léger en poids, l'affichage est dénué de pixellisation et surtout, on garde les caractères donc les mots, donc le sens des textes ce qui permet un bon positionnement. Seulement, de deux choses l'une, soit on travaille uniquement avec les polices disponibles sur le poste du visiteur, et vu que l'on ignore lesquelles il possède, on prend le minimum genre Arial, Helvetica et Times, soit on se retourne vers une solution à base de Flash à condition que le poste client puisse lire du Flash. Et sur les smartphones type iPhone, on sait que ce n'est pas idéal.
- ou bien travailler avec des images (on crée les titres dans Photoshop, et on intègre l'image qui en résulte). Cela comporte trois inconvénients majeurs : c'est lourd, difficilement référencable et totalement fermé aux handicapés.

Maintenant

Mais désormais, avec les CSS3 et la propriété « @font-face », tout cela va changer ! Le pire, c'est que cette propriété n'est pas apparue avec la CSS3, mais avec la CSS2. Seulement, en ces temps-là, d'une part vu les débits de l'Internet, embarquer une police n'était pas forcément recommandé pour un affichage rapide, mais surtout aucun navigateur en dehors d'IE (eh oui !) ne l'avait implémenté dans ses fonctions d'interprétation. Et le format proposé par IE était tellement exotique, que cela n'aurait servi à rien. Par contre, avec les nouveaux navigateurs, désormais :

- la propriété « @font-face » est désormais valide (même sous IE).
- les débits se sont franchement améliorés.
- on peut donc intégrer les polices de son choix. Seulement, cela ne veut pas dire que toutes les polices sont en libre distribution. Au contraire : comme pour les images, les polices sont régies par des droits d'auteurs qu'il faut respecter, car je peux vous assurer que la création d'une typographie est vraiment un long et difficile travail qui mérite une rétribution bien méritée. Donc, avant de mettre une police sur votre serveur pour qu'elle serve à l'affichage des textes de votre site, vérifiez les droits et mettez-vous en règle si besoin est.

Les formats de polices

Par contre, tout n'est pas devenu si simple car les navigateurs ne se sont pas mis d'accord pour tous charger le même format de polices ; cela aurait été trop simple. Du coup, si l'on veut un site fonctionnant sur tous les navigateurs, il va falloir multiplier les formats et les déclarations.

Voici les formats de polices dont il faut disposer :

- .otf (*OpenType Font*) et .ttf (*True Type Font*) : ces deux formats sont les plus utiles car reconnus par Firefox, Chrome, Safari et Opéra.
- .eot (*Embedded OpenType*) : seul format daigné être reconnu par IE.
- .svg : a sans doute de l'avenir.

- .woff : a un avenir certain.

Seulement, petit problème, car souvent le format que vous allez rencontrer en téléchargement sera le .ttf, le plus répandu. Heureusement, il existe un outil qui va vous générer tous les formats demandés à partir de ce .ttf. Il s'agit de :

@FONT-FACE GENERATOR (<http://www.fontsquirrel.com>)

Ainsi, en partant de ma police « nasaliza-webfont.ttf », j'obtiens :

- nasaliza-webfont.eot ou nasaliza-webfont.ttf
- nasaliza-webfont.svg
- nasaliza-webfont.woff

Déclaration dans la page HTML

Ensuite, indiquer la police et les formats reste un jeu d'enfant :

```
@font-face {  
    font-family: "nasaliza-webfont";  
    src: url(css/type/nasaliza-webfont.eot);  
    src: url(css/type/nasaliza-webfont.eot?#iefix)  
        format('embedded-opentype'), url(css/type/nasaliza-webfont.woff)  
        format('woff'), url(css/type/nasaliza-webfont.ttf)  
        format('truetype'), url(css/type/nasaliza-  
            webfont.svg#NasalizationMedium) format('svg');  
}
```

Résultat

Nasaliza-webfont

Exemple de texte
Taille de texte CSS (px) avec hauteur de ligne à 1.4 em

PRIAVANT LE PAPIER, SE PROMENANT DANS L'HUNDITE DE
LA NUIT REDOUTANTE, D'INFLUENCES A EXERCER.
DONNEZ-VOUS DONC LA PEINE DE PRENDRE DU VIN, ON
DETACHA LA CHALOUE POUR VOIR CE QUELLE EN SERAIT
MEURTRIE, GRAVEMENT. RACONTE...

Caractères

A B C D E F G H I J K
L M N O P Q R S T U
V W X Y Z

Pour vous faciliter l'intégration et le test de ces polices, deux outils excellents existent :

- **Web Font Specimen:** <http://wfs.typographisme.net/>
Ce site vous propose de vous générer une page avec du contenu texte, complété par ses CSS de base ; il ne vous reste plus

qu'à intégrer la web-police de votre choix, pour avoir un aperçu très complet. C'est d'ailleurs grâce à cet outil que l'image ci-contre a été générée.

► **Font Dragr:** <http://fontdragr.com/>

Remarquable outil, car il permet grâce à un widget de tester n'importe quelle police sur le site de votre choix, et cela en moins de 10 secondes. Incontournable et bluffant !

Multi-colonnage

Si vous avez un peu manipulé des contenus textes en CSS2 et HMTL4, vous devez savoir que ce n'est jamais très facile de faire du multicolonnage de manière propre et qui s'adapte aux contenus dynamiques. Il faut que les colonnes occupent tout l'espace, que leur hauteur s'adapte et qu'ils se synchronisent les uns par rapport aux autres. C'est pourquoi, les CSS3 proposent désormais une propriété qui permet de gérer sans aucun souci ce multicolonnage, avec des propriétés complémentaires qui vont agrémenter cette présentation journalistique.

Rappelons que le multicolonnage n'est pas là uniquement pour faire beau, mais bien pour faciliter la lecture de l'internaute, comme d'ailleurs dans la mise en page papier. D'ailleurs, les règles de maquette stipulent qu'une ligne de texte, pour être facilement lue et interprétée par l'œil humain et son cerveau, doit faire entre 8 et 12 mots maximum, d'où l'intérêt de scinder les textes longs en 2 ou 3 colonnes.



Attention !

Au-dessus de 3 colonnes, vous perdrez le bénéfice de cette facilité de lecture. Il faut savoir qu'un générateur de colonne performant existe qui vous simplifiera encore la tâche : <http://debray.jerome.free.fr/index.php?outils/Generateur-de-multi-colonnes-en-css3>

Mise en œuvre

Propriété : « column-count : nombre de colonnes ».



Attention !

L'emploi des préfixes propres à chaque moteur est indispensable.

Dans l'exemple ci-dessous, nous en avons profité pour rajouter les propriétés complémentaires telles que la largeur des gouttières (*gap* : écart entre les colonnes), la taille des éléments séparateurs de colonnes (*column-rule*), leur couleur (*column-rule-color*) et l'aspect de ces séparateurs (*dotted* = pointillé).

```
.multicol3 {  
    -moz-column-count:3;  
    -webkit-column-count:3;  
    -o-column-count:3;  
    column-count:3;  
    -moz-column-gap:20px;  
    -webkit-column-gap:20px;  
    -o-column-gap:20px;  
    column-gap:20px;  
    -webkit-column-rule-width:3px;  
    -webkit-column-rule-color:#cbcef8;  
    -webkit-column-rule-style:dotted;  
    -moz-column-rule-width:3px;  
    -moz-column-rule-color:#cbcef8;  
    -moz-column-rule-style:dotted;  
    -o-column-rule-width:3px;  
    -o-column-rule-color:#cbcef8;  
    -o-column-rule-style:dotted;  
    column-rule-width:3px;  
    column-rule-color:#cbcef8;  
    column-rule-style:dotted;  
}
```

Lore ipsum dolor sit amet, consectetur adipiscing elit. Praesent fringilla ultricies nisi, at pulvinar velit accumsan et. Suspendisse sem est, consectetur eu pulvinar ac, sollicitudin quis mi. In massa id, lacinia id vestibulum pellentesque, vulputate ut elit. Aenean egestas ultricies augue, vel dignissim lorem rhoncus eget. Vestibulum dui orci, laoreet ac feugiat id, facilisis eget dolor. Aliquam eu elit sapien suscipit sodales at quam. Nunc liber, eleifend nec dictum eu, luctus in odio. Ut pretium dolor sed sapien

vulputate euismod commodo nisi posuere. Vivamus ac mi velit dui vehicula rhoncus. Praesent acicul lacini consecetur. Duis ut nibh ac orci fermentum congue. Phasellus tali at lacinia velit. Cras hendrerit, arcu ac interdum adipiscing, sem neque interdum quam, at ultricies diam felis a diam. Phasellus sit amet tempor dui. Donec at urna diam. Vestibulum lobortis purus vitae metus consequat nec tindidunt metus venenatis. Nisi sit amet elit mi. Etiam mattis mauris magna, et tindidunt tindidunt ut. posuere, elit

ac bibendum suscipit, elit orci adipiscing
nisi, ac dictum maurus est ac libero.
Aenean vel nibh massa. In nulla ipsum,
egestas at consequat in facilisis eget
lacus. Aenean at est nulla. In fermentum
vestibulum purus id imperdiet. Ut
tristique, enim at ultricies iaculis, velit
massa scolitidin nibh, sit amet varius elit
nisi id ante. Curabitur metus sem, sodales
vel commodo et, semper ac lorem. Donec
tincidunt venenatis velit, eget mattis
magna nostra vel.



Animations avec les CSS3

Il est désormais possible de créer des animations entièrement avec les CSS3. Celles-ci sont séparées en deux groupes distincts :

- **les transitions** : permettent d'interpoler des propriétés CSS lorsqu'une action utilisateur est déclenchée ; cela est semblable à ce qui est possible en utilisant le JavaScript.
- **les animations** : permettent de créer des animations autonomes sans aucune action utilisateur ; elles sont semblables aux animations créées avec le logiciel Flash.

Concomitant aux animations, nous allons également découvrir les **transformations 2D**. Une contrainte liée aux animations, aux transformations et au support de celles-ci dans les navigateurs actuels est **l'utilisation des préfixes**.

Les propriétés CSS officielles existent mais ce sont les versions préfixées qui dominent à l'heure actuelle.

Les transitions

Une transition est une **interpolation entre deux valeurs d'une propriété CSS**. De ce fait, les transitions ne peuvent s'appliquer que sur les propriétés CSS dont la **valeur est numérique**. En effet, il est difficile d'interpoler le mot « relative » vers « absolute », mais il est plus naturel d'interpoler un corps de texte ou une largeur de bordure.

Les propriétés CSS interpolables

La liste des propriétés CSS interpolables est disponible à l'adresse :

<http://www.w3.org/TR/css3-transitions/#animatable-properties>.

Les déclencheurs

Une transition aura forcément **un point initial et un point final**. Les propriétés CSS définies sur l'élément interpolé sont considérées comme les valeurs initiales de la transition. Les valeurs finales seront, quant à elles, contenues dans une **pseudo-classe CSS** qui sera considérée comme le **déclencheur de la transition**.

Un déclencheur peut représenter une action de l'utilisateur comme les pseudo-classes **hover** ou **focus** ou une action du navigateur comme **valid** ou **required**.

La propriété raccourcie transition

HTML

```
<div id="carre"></div>
```

CSS

```
#carre{  
width:100px;  
height:100px;  
background:hsl(50, 100%, 80%);  
-moz-transition:background 1s ease-in 0s;  
-o-transition:background 1s ease-in 0s;  
-webkit-transition:background 1s ease-in 0s;  
-ms-transition:background 1s ease-in 0s;  
transition:background 1s ease-in 0s;  
}  
#carre:hover{  
background:hsl(50, 100%, 50%);  
}
```

La propriété CSS *transition* va vous permettre de **créer et paramétrier une transition**. Cette propriété peut s'écrire de manière raccourcie, c'est-à-dire en n'utilisant que la seule propriété CSS

transition pour créer et paramétrier l'interpolation. Elle contient quatre paramètres à utiliser dans un ordre strict :

- **liste des propriétés à interpoler** : séparées par des virgules, ou en utilisant le mot-clé *all* pour interpoler toutes les propriétés.
- **durée de l'interpolation** : en utilisant l'unité de mesure « *s* » pour une durée exprimée en secondes, ou « *ms* » pour l'utilisation des millisecondes.
- **effet de l'interpolation** : la liste est décrite ci-après.
- **délai avant le déclenchement de l'interpolation** : exprimé en secondes ou millisecondes.

Seuls les deux premiers paramètres sont obligatoires. Dans notre exemple, la transition se fait sur la propriété *background*, dure une seconde, avec un effet d'accélération et sans délai avant le déclenchement de l'interpolation. Le déclencheur de l'interpolation est le survol de l'élément grâce à l'utilisation de la pseudo-classe **hover**.

À l'heure où nous écrivons ces lignes, les transitions ne sont pas prises en charge par Internet Explorer 9 mais elles le seront par la version 10, d'où l'utilisation du préfixe « *-ms-* » dans notre exemple.



Attention !

Pour éviter de surcharger ce livre, nous utiliserons désormais la propriété CSS transition officielle pour nos exemples, en sachant qu'il faut absolument utiliser les préfixes dans un cas réel.

Les propriétés uniques

CSS

```
transition-property:background;  
transition-duration:1s;  
transition-timing-function:ease-in;  
transition-delay:0s;
```

Chaque paramètre de la propriété raccourcie *transition* peut être écrit avec sa propriété unique. Ces propriétés sont :

- **transition-property** : liste des propriétés à interpoler.

- **transition-duration** : durée de l'interpolation.
- **transition-timing-function** : effet de l'interpolation.
- **transition-delay** : délai avant le déclenchement de l'interpolation.

L'utilisation des propriétés uniques n'est pas recommandée car il faut également préfixer chacune d'elles, c'est-à-dire qu'il faudrait une vingtaine de lignes pour avoir le même résultat qu'en utilisant cinq lignes avec la propriété raccourcie.

Les effets de l'interpolation

Il est possible de donner un effet à l'interpolation pour améliorer l'impact visuel de la transition. Les effets disponibles sont :

- **linear** : mouvement linéaire.
- **ease** : décélération accentuée, c'est la valeur par défaut.
- **ease-in** : accélération.
- **ease-out** : décélération.
- **ease-in-out** : accélération puis décélération.
- **steps** : accélération par paliers.
- **cubic-bezier** : mouvement personnalisé.

Seuls les effets steps et cubic-bezier sont paramétrables.

□ L'effet steps

CSS

```
transition:background 3s steps(3, end) 0s;
```

L'effet **steps** est une interpolation par paliers, c'est-à-dire que l'interpolation n'est pas fluide mais s'exécute par à-coups. Cet effet peut recevoir deux paramètres mais seul le premier est obligatoire :

- **le nombre de paliers** : sous forme d'un chiffre entier.
- **modification de la propriété interpolée** : avec l'utilisation des mots-clés *start* si la propriété doit être modifiée au début de chaque palier, ou *end* si la propriété doit être modifiée à la fin de chaque palier.

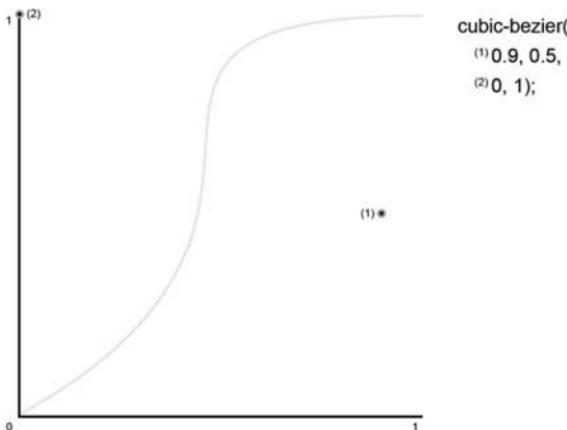
En utilisant *start*, l'interpolation se déclenche sans délai, alors qu'avec *end*, l'interpolation se déclenche après un délai égal à la durée de la transition divisée par le nombre de paliers.

□ L'effet cubic-bezier

CSS

```
transition:background 3s cubic-bezier(0.9, 0.5, 0, 1);
```

L'effet cubic-bezier est la **représentation d'une courbe de Bézier**.



La courbe est définie dans un **plan cartésien de 1 sur 1** avec deux points de contrôle. Pour chacun d'eux, il faudra définir sa position en abscisse et en ordonnée. Ces positions seront des chiffres décimaux sur base 1. Il est possible d'utiliser des chiffres supérieurs à 1 ou inférieurs à 0 pour obtenir des effets de rebond, mais cela n'est pas pris en charge par la version actuelle du navigateur Safari.

L'effet cubic-bezier nécessite donc quatre paramètres :

- position x du premier point de contrôle ;
- position y du premier point de contrôle ;
- position x du second point de contrôle ;
- position y du second point de contrôle.

Les interpolations multiples

HTML

```
<div id="carre"></div>
```

CSS

```
#carre{  
width:100px;  
height:100px;  
background:hsl(50, 100%, 80%);  
transition:background 1s ease-in, width 0.5s ease-in-out, height 2s  
ease;  
}  
#carre:hover{  
background:hsl(50, 100%, 50%);  
width:150px;  
height:120px;  
}
```

Il est également possible de configurer la **transition de plusieurs propriétés CSS avec des valeurs, des durées ou des effets différents.**

En utilisant la propriété raccourcie, il faut séparer chaque propriété interpolée par des virgules.

CSS

```
transition-property:background, width, height;  
transition-duration:1s, 0.5s, 2s;  
transition-timing-function:ease-in, ease-in-out, ease;
```

La séparation des valeurs par des virgules est obligatoire si vous souhaitez utiliser les propriétés uniques de *transition*. De plus, l'ordre des propriétés CSS interpolées doit être identique dans chacune des propriétés uniques.

Les interpolations en chaîne

CSS

```
transition:  
background 1s ease-in 0s,  
width 0.5s ease-in-out 1s,  
height 2s ease 1.5s;
```

Pour créer des interpolations en chaîne, il faudra essentiellement configurer le **délai avant le déclenchement de l'interpolation**. Chaque interpolation tiendra compte de la durée des interpolations qui la précède.

Dans notre exemple, l'interpolation sur la propriété *background* se déclenche sans délai et dure une seconde. Puis, la propriété *width* est interpolée pendant une demi-seconde mais se déclenche après un délai d'une seconde, qui correspond à la durée de l'interpolation de la propriété *background*. Enfin, la propriété *height* dure deux secondes et son délai de déclenchement est également l'addition de la durée des interpolations qui la précède, soit une seconde et demie.

CSS

```
transition-property:background, width, height;  
transition-duration:1s, 0.5s, 2s;  
transition-timing-function:ease-in, ease-in-out, ease;  
transition-delay:0s, 1s, 1.5s;
```

Les animations

Flash et le JavaScript n'ont qu'à bien se tenir !

Les CSS3 permettent désormais d'**animer des éléments HTML** sans aucune utilisation de Flash ou de JavaScript. Pour animer avec les CSS3, il est nécessaire d'utiliser, d'une part la règle CSS **@keyframes**, et d'autre part la propriété CSS *animation*. Enfin, les animations ne sont pas prises en charge actuellement par les navigateurs Internet Explorer 9 – mais le seront par Internet Explorer 10 – et Opera.

La règle CSS @keyframes

CSS

```
@-moz-keyframes monAnimation{  
0%{width:100px;}  
100%{width:200px;}  
}  
@-o-keyframes monAnimation{  
0%{width:100px;}
```

```
100%{width:200px;}  
}  
@-webkit-keyframes monAnimation{  
0%{width:100px;}  
100%{width:200px;}  
}  
@-ms-keyframes monAnimation{  
0%{width:100px;}  
100%{width:200px;}  
}  
@keyframes monAnimation{  
0%{width:100px;}  
100%{width:200px;}  
}
```

Avant d'animer un élément HTML, il va falloir définir une **règle CSS @keyframes** qui va servir à paramétriser l'animation. Cette règle CSS crée une **ligne temporelle virtuelle** qui débute à la valeur **0 %** pour se terminer à la valeur **100 %**. Il est à noter que la valeur **0 %** peut être remplacée par la valeur **from**, et la valeur **100 %** par la valeur **to**.

L'animation va comporter un certain nombre d'**images-clés sous forme de pourcentages**, qui sont des moments où les propriétés CSS de l'élément animé vont être modifiées. Ces pourcentages ne sont que des valeurs relatives par rapport à la durée de l'animation que vous allez définir avec la propriété CSS *animation*. Par exemple, si vous définissez la durée de l'animation à cinq secondes, la valeur **0 %** va correspondre au démarrage de l'animation, la valeur **30 %** va correspondre à une seconde et demie, et la valeur **80 %** à la quatrième seconde.

Il est également indispensable de **nommer l'animation** pour ensuite utiliser ce nom sur l'élément HTML à animer. Ce nom ne peut comporter ni d'espace, ni de caractères spéciaux sauf le tiret et le tiret bas, ni débuter par un chiffre.

Dans notre exemple :

- l'animation est nommée **monAnimation** ;
- l'animation est composée de **deux images-clés** à **0 %** et à **100 %** ;
- la règle CSS **@keyframes** doit être utilisée avec **les préfixes de chaque navigateur**.

La propriété raccourcie animation

HTML

```
<div id="carre"></div>
```

CSS

```
#carre{  
width:100px;  
height:100px;  
background:hsl(50, 100%, 80%);  
-moz-animation:monAnimation 3s ease 0s infinite alternate;  
-o-animation:monAnimation 3s ease 0s infinite alternate;  
-webkit-animation:monAnimation 3s ease 0s infinite alternate;  
-ms-animation:monAnimation 3s ease 0s infinite alternate;  
animation:monAnimation 3s ease 0s infinite alternate;  
}
```

La propriété CSS *animation* permet d'affecter une animation créée avec la règle CSS @keyframes à un élément HTML. Elle peut recevoir pas moins de six paramètres, dont seuls les trois premiers sont obligatoires :

- **nom de l'animation** : défini dans la règle CSS @keyframes.
- **durée de l'animation** : en secondes ou millisecondes.
- **effet de l'animation** : identique aux effets utilisés pour la propriété CSS *transition*.
- **délai avant le déclenchement de l'animation** : en secondes ou millisecondes.
- **nombre d'itérations de l'animation** : par défaut, la valeur est de 1, c'est-à-dire que l'animation ne boucle pas ; les valeurs possibles sont, soit un *chiffre entier* représentant le nombre total d'itérations, soit le mot-clé *infinite* qui signifie que l'animation tourne en boucle.
- **sens de lecture de l'animation** : les valeurs possibles sont, soit le mot-clé *normal*, qui est la valeur par défaut, et qui signifie que l'animation se jouera toujours de sa première image-clé vers sa dernière image-clé, soit le mot-clé *alternate* qui donne à l'animation un effet de yoyo, c'est-à-dire que l'animation va jouer de sa première image-clé vers sa dernière image, puis va faire le chemin inverse, de sa dernière image-clé vers sa première image-clé.



Attention !

Pour éviter de surcharger ce livre, nous utiliserons désormais la règle CSS @keyframes et la propriété CSS animation officielle pour nos exemples, en sachant qu'il faut absolument utiliser les préfixes dans un cas réel.

Les propriétés uniques

CSS

```
animation-name:monAnimation;  
animation-duration:3s;  
animation-timing-function:ease;  
animation-delay:0s;  
animation-iteration-count:infinite;  
animation-direction:alternate;
```

Chaque paramètre de la propriété raccourcie *animation* peut être écrit avec sa propriété unique. Ces propriétés sont :

- **animation-name** : nom de l'animation.
- **animation-duration** : durée de l'animation.
- **animation-timing-function** : effet de l'animation.
- **animation-delay** : délai avant le déclenchement de l'animation.
- **animation-iteration-count** : nombre d'itérations de l'animation.
- **animation-direction** : sens de lecture de l'animation.

L'utilisation des propriétés uniques n'est pas recommandée car il faut également préfixer chacune de ces propriétés.

Les animations multiples

HTML

```
<div id="carre"></div>
```

CSS

```
@keyframes monAnimation{  
0%{width:100px;}
```

```
100%{width:200px; }
}
@keyframes secondeAnimation{
0%{height:100px; opacity:1;}
50%{opacity:0;}
100%{height:200px; opacity:1;}
}
#carre{
width:100px;
height:100px;
background:hsl(50, 100%, 80%);
animation:monAnimation 3s ease, secondeAnimation 0.5s ease-out;
}
```

Il est possible de créer différentes animations applicables à un même élément, ce qui permet d'**animer des propriétés CSS avec des valeurs, des durées et des effets différents**. Avec la propriété raccourcie, chaque animation sera séparée par des virgules.

Dans notre exemple, nous avons créé deux animations, nommées respectivement **monAnimation** et **secondeAnimation** possédant des durées et des effets différents.

La propriété **animation-fill-mode**

La propriété **animation-fill-mode** ne fait pas partie de la propriété raccourcie *animation*. Cependant, elle est très utile car elle sert à **définir quels styles CSS sont appliqués à l'élément animé avant et/ou après l'animation**. Si votre animation ne tourne pas en boucle grâce à la propriété *animation-iteration-count* définie sur la valeur *infinite*, l'élément retrouve ses styles CSS initiaux dès la fin de l'animation.

Plusieurs valeurs sont possibles :

- **none** : l'animation n'affecte aucun style CSS à l'élément animé, c'est-à-dire que les propriétés CSS définies dans les images-clés de l'animation ne seront affectées à l'élément ni avant ou ni après l'animation, c'est la valeur par défaut.
- **backwards** : les propriétés CSS définies à la première image-clé de l'animation, soit **0 %**, soit **from**, sont affectées à l'élément animé dès le début de l'animation.

- **forwards** : les propriétés CSS définies à la dernière image-clé de l'animation, soit **100 %**, soit **to**, sont affectées à l'élément animé après la fin de l'animation.
- **both** : rassemble les spécificités des valeurs **backwards** et **forwards**.

Les transformations 2D

Souvenez-vous des nombreux moments où vous vous êtes dit : « Ce serait possible d'appliquer une rotation à mon image ? », la réponse était : « C'est impossible sur le Web ! ». C'est désormais possible et même bien plus. En effet, les CSS3 nous apportent enfin la possibilité de **modifier l'apparence de n'importe quelle balise HTML**.

Ces transformations 2D sont accompagnées de transformations 3D qui ne sont peu ou pas encore pris en charge par les navigateurs à l'heure où nous écrivons ces lignes. Tout comme les transitions et les animations, il est encore nécessaire de **préfixer les propriétés CSS de transformation**.

La propriété raccourcie transform

La propriété CSS *transform* va contenir toutes les transformations que vous souhaitez appliquer à un élément. Celles-ci ne sont **pas séparées par des virgules**. L'ordre des transformations est également important car les valeurs de chacune d'elles dépendent des transformations précédentes. Enfin, par défaut, le point de référence des transformations est le centre de l'élément.

Le déplacement

HTML

```
<div id="carre"></div>
```

CSS

```
#carre{  
-moz-transform:translate(50px, 70px);  
-o-transform:translate(50px, 70px);  
-webkit-transform:translate(50px, 70px);  
}
```

```
-ms-transform:translate(50px, 70px);  
transform:translate(50px, 70px);  
}
```

L'instruction *translate* permet d'effectuer un déplacement de l'élément HTML. Elle peut être appliquée sous trois formes différentes :

- **translate** : nécessitant deux paramètres, la valeur du déplacement en abscisse et en ordonnée.
- **translateX et translateY** : nécessitant un seul paramètre, qui est la valeur du déplacement soit en abscisse, soit en ordonnée.

Toutes les **unités de mesure CSS existantes** peuvent être utilisées pour définir la valeur de chaque décalage.

CSS

```
transform:translateX(50px) translateY(70px);
```

La transformation effectuée ci-dessus est l'équivalent de l'exemple ci-haut, mais en utilisant les instructions *translateX* et *translateY*.



Attention !

Pour éviter de surcharger ce livre, nous utiliserons désormais la propriété CSS transform officielle pour nos exemples, en sachant qu'il faut absolument utiliser les préfixes dans un cas réel.

La rotation

CSS

```
transform:rotate(45deg);  
transform:rotate(-0.78rad);
```

Pour appliquer une rotation à un élément HTML, il faudra utiliser l'instruction *rotate*.

Il est possible de spécifier deux types d'unités de mesure :

- **les degrés** : en utilisant le mot-clé *deg*.
- **les radians** : en utilisant le mot-clé *rad*.

Si vous souhaitez appliquer une **rotation dans le sens inverse des aiguilles d'une montre**, il faudra utiliser une valeur négative.

La mise à l'échelle

CSS

```
transform:scale(-1.2, 0.5);
```

La mise à l'échelle d'un élément HTML est possible grâce à l'instruction *scale*. Il existe quatre moyens d'appliquer une mise à l'échelle:

- **scale avec deux paramètres** : les paramètres seront respectivement l'échelle horizontale et l'échelle verticale.
- **scale avec un paramètre** : l'échelle sera identique en horizontal et en vertical.
- **scaleX et scaleY** : nécessitent un seul paramètre, qui est la valeur de l'échelle, soit en horizontal, soit en vertical.

La valeur de l'échelle est un **chiffre décimal sur base 1**. La valeur 1 représente 100 %, c'est-à-dire l'élément à sa taille originelle. Des **valeurs négatives sont possibles** et auront pour effet de renverser l'élément.

CSS

```
transform:scaleX(-1.2) scaleY(0.5);
```

La transformation effectuée ci-dessus est l'équivalent de l'exemple ci-haut, mais en utilisant les instructions *scaleX* et *scaleY*.

L'inclinaison

CSS

```
transform:skewX(10deg) skewY(-0.5rad);
```

Incliner un élément HTML est possible en utilisant les instructions *skewX* et *skewY*. L'instruction *skewX* permet de spécifier un **angle d'inclinaison** sur l'axe horizontal, et *skewY* sur l'axe vertical. Tout comme l'instruction *rotate*, les angles peuvent être exprimés en degrés ou en radians, et posséder une valeur positive ou négative.

Une matrice de transformation

CSS

```
transform:matrix(0.7, -0.7, 0.7, 0.7, 0, 0);
```

Si vous souhaitez **combiner plusieurs transformations** en une seule instruction, il est préférable d'utiliser une matrice de transformation avec l'instruction *matrix*, mais son utilisation n'est pas aisée.

Une matrice de transformation nécessite six paramètres. Chacun d'eux est le résultat d'un calcul mathématique prenant en compte la valeur de toutes les transformations.

La matrice (1, 0, 0, 1, 0, 0) ne fait subir aucune transformation à l'élément.

Un outil pour faciliter l'utilisation d'une matrice de transformation

Consultez la page : <http://peterned.home.xs4all.nl/matrices/>

La propriété transform-origin

CSS

```
-moz-transform-origin:bottom right;
```

La propriété *transform-origin* permet de **modifier le point de référence** d'une transformation. Les valeurs à utiliser sont :

- **des chiffres entiers** : représentant des positions fixes par rapport au coin supérieur gauche.
- **des pourcentages** : représentant des valeurs relatives par rapport aux dimensions de l'élément, et avec comme point de référence, le coin supérieur gauche.
- **des mots-clés** : *top* pour le bord haut, *bottom* pour le bord bas, *left* pour le bord gauche, *right* pour le bord droit et *center* pour le centre.

Pour les chiffres entiers et les pourcentages, **l'ordre des valeurs est strict** :

- la première est la valeur en abscisse ;

- la seconde est la valeur en ordonnée.

Pour les mots-clés, si un seul est renseigné, le second prendra la valeur *center*. Il est possible d'utiliser des **valeurs négatives** pour que le point de référence soit à l'extérieur de l'élément.

La propriété *transform-origin* a besoin d'être **préfixée** pour être fonctionnelle dans les navigateurs actuels.

Les transitions, les animations et les transformations

Pour utiliser une transformation dans une transition ou une animation, il faudra toujours utiliser les **préfixes pour chaque navigateur**.

□ Une transition avec transformation

HTML

```
<div id="carre"></div>
```

CSS

```
#carre{  
width:100px;  
height:100px;  
background:hsl(50, 100%, 80%);  
-moz-transition:-moz-transform 0.3s ease-out;  
-o-transition:-o-transform 0.3s ease-out;  
-webkit-transition:-webkit-transform 0.3s ease-out;  
-ms-transition:-ms-transform 0.3s ease-out;  
transition:transform 0.3s ease-out;  
}  
  
#carre:hover{  
-moz-transform:scaleX(1.1) scaleY(1.1);  
-o-transform:scaleX(1.1) scaleY(1.1);  
-webkit-transform:scaleX(1.1) scaleY(1.1);  
-ms-transform:scaleX(1.1) scaleY(1.1);  
transform:scaleX(1.1) scaleY(1.1);  
}
```

□ Une animation avec transformation

CSS

```
@-moz-keyframes monAnimation{  
0%{-moz-transform:scaleX(1) scaleY(1);}  
100%{-moz-transform:scaleX(1.1) scaleY(1.1);}  
}
```

```
@-o-keyframes monAnimation{  
0%{-o-transform:scaleX(1) scaleY(1);}  
100%{-o-transform:scaleX(1.1) scaleY(1.1);}  
}  
@-webkit-keyframes monAnimation{  
0%{-webkit-transform:scaleX(1) scaleY(1);}  
100%{-webkit-transform:scaleX(1.1) scaleY(1.1);}  
}  
@-ms-keyframes monAnimation{  
0%{-ms-transform:scaleX(1) scaleY(1);}  
100%{-ms-transform:scaleX(1.1) scaleY(1.1);}  
}  
@keyframes monAnimation{  
0%{transform:scaleX(1) scaleY(1);}  
100%{transform:scaleX(1.1) scaleY(1.1);}  
}  
#carre{  
-moz-animation:monAnimation 1s ease-out 0s infinite alternate;  
-o-animation:monAnimation 1s ease-out 0s infinite alternate;  
-webkit-animation:monAnimation 1s ease-out 0s infinite alternate;  
-ms-animation:monAnimation 1s ease-out 0s infinite alternate;  
animation:monAnimation 1s ease-out 0s infinite alternate;  
}
```




CSS3 pour mobiles et tablettes

La conception multi-écrans est devenue une réalité pour les **concepteurs Web d'aujourd'hui**. L'accès au Web, autrefois réservé aux seuls ordinateurs de bureau, s'étend aujourd'hui aux mobiles, aux consoles de jeu portables, aux télévisions, bientôt à notre réfrigérateur... **La consultation multi-écrans nécessite une conception multi-écrans.**

Chaque support de consultation possède des caractéristiques physiques, comme sa largeur et sa hauteur, des caractéristiques techniques comme sa résolution ou son format d'écran, qui lui sont propres.

Les **requêtes de média** (*media queries* en anglais) permettent de restreindre les feuilles de styles à appliquer selon les caractéristiques du support de consultation.

L'ensemble des pratiques destinées à rendre un site Web adapté à chaque écran se désigne sous le vocable « **design adaptatif** » (*responsive design* en anglais), dont l'article de référence a été écrit sur le site Web « *A List Apart* ».

L'espace visualisable

Les dimensions d'un support mobile

Une des premières interrogations que se pose un concepteur Web est la **largeur d'écran** de son futur internaute. Il va se baser sur une largeur standard pour créer la maquette du site Web.

Sur un écran bureau, l'espace disponible est abondant et la majorité des sites Web ont une largeur fixe, qui varie généralement entre 900 pixels et 1 000 pixels, et sont centrés dans le navigateur.

Sur un mobile ou une tablette, l'espace est précieux et il est nécessaire de couvrir la totalité de l'espace disponible.

Le premier problème d'un support mobile – téléphone mobile ou tablette – est qu'il est possible de se baser sur trois dimensions différentes :

- **les dimensions de l'espace visualisable** (*viewport* en anglais), définies par le navigateur.
- **les dimensions du support de visualisation** (*device* en anglais), définies par l'écran entier.
- **la résolution de l'écran.**

Aussi surprenant que cela puisse paraître, **ces trois dimensions ne possèdent pas de valeurs égales**. Si vous possédez un téléphone intelligent, vous avez remarqué que sur les sites non optimisés pour les mobiles, le site apparaît minuscule. En réalité, chaque navigateur mobile a défini des dimensions pour l'espace visualisable, différentes non seulement de celles du support de visualisation mais aussi de la résolution de l'écran. Ainsi, sur un support mobile, le site consulté s'adapte automatiquement aux dimensions de l'espace visualisable.

Quand les pixels nous jouent des tours

Pour s'apercevoir des difficultés à déterminer l'espace visualisable, prenons l'exemple de l'iPhone :

- ▶ l'iPhone 3 possède une résolution d'écran de 320 x 480 pixels, des dimensions du support également de 320 x 480 pixels, et un espace visualisable de 980 pixels en largeur.
- ▶ l'iPhone 4 possède une résolution d'écran de 640 x 960 pixels, mais des dimensions du support de 320 x 480 pixels, et un espace visualisable de 980 pixels en largeur.

Garcimore travaille-t-il chez Apple ? Comment un écran qui n'a pas changé de taille peut-il avoir une résolution doublée ? **La réponse est qu'un pixel n'est pas un pixel sur les supports mobiles.**

Sur l'iPhone4, un pixel est composé d'un carré de 2 x 2 pixels, soit 4 pixels, d'où la résolution doublée mais pas la taille de l'écran. On nomme cette propriété d'un écran comme la **densité de pixels**. Ce doublement des pixels est possible grâce au fameux **écran Retina** d'Apple, également disponible sur l'iPad 3.

Ainsi, l'iPhone 3 possède une densité de pixels de 163 points par pouce (*dots per inch* ou dpi en anglais), et l'iPhone 4 de 326 points par pouce.

La balise `<meta> viewport`

HTML

```
<meta name="viewport" content="..." />
```

Pour éviter de perdre des cheveux avec tous ces paramètres, il est fortement recommandé d'utiliser la balise **<meta> viewport**, grâce à laquelle vous allez préciser les dimensions de l'espace visualisable, la valeur du zoom effectué par le navigateur ou l'utilisateur, et la densité de pixels du support.

Comme valeur de l'attribut *content*, il est possible d'utiliser plusieurs conditions, séparées par des virgules :

- ▶ **width et height** : déterminent respectivement la largeur et la hauteur de l'espace visualisable ; les valeurs possibles sont, soit un *chiffre entier* en pixels, soit une conformité avec les dimensions du support de visualisation grâce aux mots-clés *device-width* ou *device-height*.
- ▶ **initial-scale** : spécifie la valeur de zoom initial effectué par le navigateur ; la valeur est un *chiffre décimal* sur base 1.

- **minimum-scale et maximum-scale** : spécifient respectivement les valeurs de zoom minimum et maximum que l'internaute est autorisé à effectuer ; les valeurs sont également des *chiffres décimaux* sur base 1.
- **user-scalable** : précise la possibilité par l'internaute de zoomer ; ses valeurs sont, soit *yes*, soit *no*.
- **target-densitydpi** : spécifie la densité de pixels du support de visualisation ; les valeurs possibles sont, soit un *chiffre entier*, soit une conformité avec la densité de pixels du support de visualisation avec le mot-clé *device-dpi*, soit le mot-clé *low-dpi* qui correspond à une densité de 120 points par pouce, soit le mot-clé *medium-dpi* qui correspond à une densité de 160 points par pouce, c'est la valeur par défaut, soit le mot-clé *high-dpi* qui correspond à une densité de 240 points par pouce.

HTML

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no" />
```

L'exemple ci-dessus signifie que l'espace visualisable doit se conformer à la largeur du support de visualisation, que le navigateur doit afficher la page à 100 % lors de son chargement et que l'internaute ne pourra pas zoomer.

HTML

```
<meta name="viewport" content="width=device-width, minimum-scale=1.0, target-densitydpi=326dpi" />
```

Cet autre exemple détermine que l'espace visualisable doit se conformer à la largeur du support de visualisation, que le navigateur ne pourra zoomer au-dessous de 100 % et que la densité de pixels ciblée est de 326 points par pouce.

Les requêtes de média

Après avoir défini l'espace visualisable, il est désormais possible, grâce aux CSS3, de cibler un média précis à l'aide d'une **requête de média** qui nécessite des paramètres identiques :

- le ciblage d'un **type de média** ;

- des **opérateurs logiques** comme *and*, *ou* symbolisé par une virgule, *only* ou *not* ;
- des **conditions** écrites entre parenthèses et décrivant les caractéristiques physiques et techniques du support de visualisation ciblé.

Les différentes méthodes d'importation

Pour préciser ces différents paramètres, **trois méthodes d'importation** des CSS contenant des requêtes de média sont possibles :

- la requête est écrite dans la balise `<link>` grâce à l'attribut *media*.
- l'instruction CSS `@import`, nécessitant plusieurs fichiers CSS et contenant la requête de média.
- l'instruction CSS `@media`, écrite dans un fichier CSS unique.

L'attribut media de la balise `<link>`

HTML

```
<link type="text/css" rel="stylesheet" href="requeteMedia.css"  
      media="type de média and (condition)" />
```

CSS

```
requeteMedia.css  
règles CSS
```

L'utilisation de la balise `<link>` semble la plus naturelle pour contenir les requêtes de média. Avec cette méthode d'importation, le fichier CSS n'est chargé que si le type de média est celui utilisé par l'internaute, et si les conditions correspondent aux caractéristiques du support de visualisation. L'avantage de cette méthode est la répartition des règles CSS dans des fichiers uniques ciblant un média unique.

L'instruction CSS `@import`

HTML

```
<link type="text/css" rel="stylesheet" href="requeteMedia01.css" />
```

CSS

```
requeteMedia01.css
```

```
@import url("requeteMedia02.css") type de média and (condition);
requeteMedia02.css
règles CSS
```

La méthode d'importation utilisant l'instruction `@import` nécessite au minimum deux fichiers CSS.

Le fichier CSS appelé par la balise `<link>` va jouer le rôle d'aiguilleur, en répartissant l'importation des fichiers CSS selon le type de média et les conditions spécifiées.

□ L'instruction CSS `@media`

HTML

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

CSS

```
requeteMedia.css
@media type de média and (condition){
règles CSS
}
```

L'instruction CSS `@media` est caractérisée par le fait que les requêtes de média sont intégrées parmi les autres règles CSS. Cette méthode peut être recommandée lorsque les règles spécifiques à un média sont peu nombreuses, mais elle devient vite illisible lors de son utilisation avec un site Web d'importance.

Les types de médias

Le W3C a défini, dès la spécification des CSS2, une dizaine de types de médias, mais beaucoup d'entre eux ne sont toujours pas pris en charge par les appareils actuels.

TYPES DE MÉDIAS CSS2

Type de média	Description du média
all	Tous types de médias
braille	Média braille à retour tactile
embossed	Média à impression braille
handheld	Média portatif de petite taille avec écran monochrome

Type de média	Description du média
print	Support paginé et documents écran prêts à l'impression
projection	Présentations en projection
screen	Média avec moniteur couleur
speech	Synthétiseurs de parole
tty	Média utilisant une grille de caractères à chasse fixe
tv	Média de type télévision

Concernant les mobiles et les tablettes, le type de média *handheld* paraît le plus adapté, mais en réalité celui-ci est très peu pris en charge par les navigateurs mobiles. *A contrario*, le type *screen* est plus générique et permet tout autant de cibler les mobiles que les tablettes ou encore les écrans d'ordinateurs de bureau.

Les caractéristiques du support de visualisation

Un ensemble de critères vont nous permettre de cibler très précisément un type de média grâce à ses **caractéristiques physiques et techniques**.

□ Les caractéristiques physiques

Elles sont accessibles à travers les conditions suivantes :

- **width et height** : largeur et hauteur de l'espace visualisable ; la valeur est un *chiffre entier*, généralement exprimé en pixels dans le cas des mobiles et des tablettes.
- **device-width et device-height** : largeur et hauteur du support de visualisation ; la valeur est également un *chiffre entier*.
- **orientation** : orientation du support de visualisation ; les valeurs possibles sont les mots-clés *portrait* (vertical) ou *landscape* (horizontal).
- **aspect-ratio et device-aspect-ratio** : format d'affichage de l'espace visualisable et support de visualisation ; la valeur est une *fraction* qui peut être, soit la proportion de l'image, par exemple 16/9 ou 4/3, soit la résolution de l'écran, par exemple 1 280/720 ou 1 024/768.

□ Les caractéristiques techniques

- **color** : support de visualisation utilisant un affichage en couleur ; la valeur est un *chiffre entier* représentant le nombre de bits par couleur.
- **color-index** : support de visualisation utilisant un affichage avec une table de couleurs indexées ; la valeur est un *chiffre entier* représentant le nombre de couleurs contenues dans la table.
- **monochrome** : support de visualisation utilisant un affichage en monochrome ou en niveaux de gris ; la valeur est un *chiffre entier* représentant le nombre de bits par pixel.
- **resolution** : résolution du support de visualisation ; la valeur est un *chiffre entier* exprimé, soit en points par pouce (dpi), soit en points par centimètre (dpcm).
- **scan** : méthode de balayage des écrans de télévision ; les valeurs possibles sont les mots-clés *progressive* (balayage progressif) ou *interlace* (balayage entrelacé).
- **grid** : support de visualisation utilisant affichage en grille ou matricielle (*bitmap* en anglais) ; la valeur est *booléenne*, 1 signifie que l'affichage est en grille, 0 signifie que l'affichage est matriciel.
- **webkit-min-device-pixel-ratio** : densité de pixels ; la valeur est un *chiffre entier* représentant la densité de pixels du support de visualisation, ce paramètre n'est reconnu que par les navigateurs mobiles utilisant le moteur de rendu HTML WebKit, dont Chrome sur Android et Safari sur iOS.

□ Les préfixes min et max

Toutes les conditions, à l'exception de **orientation**, **scan** et **grid**, acceptent les **préfixes min et max**. Ces derniers représentent respectivement la valeur minimale et maximale d'une condition, et sont indispensables pour améliorer le ciblage des supports de visualisation. Il est donc possible d'utiliser les conditions **min-width**, **max-device-width** ou encore **min-color**.

Exemples de requêtes de média

Pour présenter les exemples, nous allons utiliser la méthode d'importation des requêtes de média utilisant l'instruction CSS **@media**. Nous nous intéressons principalement aux **mobiles** et

aux tablettes, d'où le type de médias qui sera toujours *screen*. Enfin, nous considérons que les dimensions de l'espace visualisable sont conformes aux dimensions du support de visualisation grâce à la balise **meta viewport**.

CSS

```
@media only screen and (color)
```

Il est possible d'utiliser des conditions sans spécifier leur valeur. Cet exemple cible exclusivement tous les écrans couleurs.

CSS

```
@media only screen  
and (orientation : landscape)  
and (min-width : 320px)  
and (min-color : 2)  
and (device-aspect-ratio : 16/9)
```

Une condition n'accepte qu'une seule valeur. Cet exemple cible les écrans avec une orientation horizontale, une largeur minimale d'espace visualisable de 320 pixels, affichant au minimum 2 couleurs, et dont le format d'affichage du support de visualisation est le 16/9^e.

CSS

```
@media screen and (color)  
and (min-width : 320px)  
and (max-width : 480px)
```

Cet exemple permet de cibler l'iPhone, car le support possède des dimensions au minimum de 320 pixels lorsqu'il est en position verticale, et d'un maximum de 480 pixels lorsque celui-ci est en position horizontale.

CSS

```
@media screen and (color)  
and (width : 768px)  
and (height : 1024px)  
and (orientation : portrait)
```

Cet exemple permet de cibler uniquement l'iPad en position verticale... Un peu trop spécifique, vous ne trouvez pas ?

Cas pratique

Pour illustrer les requêtes de média, rien ne vaut un cas pratique. Nous allons définir **trois groupes de support de visualisation** :

- les ordinateurs de bureau et les portables disposant d'un large écran ;
- les tablettes de type iPad et les netbooks ;
- les mobiles intelligents de type iPhone.

Chaque groupe va disposer de sa propre mise en page avec quelques contraintes :

- les **ordinateurs de bureau et portables à large écran** disposeront d'une maquette à largeur fixe sur deux colonnes et centrée dans le navigateur.
- les **tablettes et netbooks** disposeront d'une maquette sur deux colonnes à largeur variable, et également des images qui se redimensionnent.
- les **mobiles intelligents** disposeront d'une maquette sans colonne et avec des images qui se redimensionnent selon la largeur de l'écran.

Les largeurs utilisées seront :

- pour les ordinateurs de bureau et portables à large écran : plus de 1 024 pixels.
- pour les tablettes et les netbooks : de 480 pixels minimum à 1 024 pixels maximum.
- pour les mobiles intelligents : 480 pixels maximum.

Mise en place du HTML

HTML

```
<!-- contenu -->
<div id="conteneur">

<!-- en-tête de page -->
<header>header</header>

<!-- menu -->
<nav>
```

```
<ul>
<li><a href="#">rubrique1</a></li>
<li><a href="#">rubrique2</a></li>
<li><a href="#">rubrique3</a></li>
<li><a href="#">rubrique4</a></li>
<li><a href="#">rubrique5</a></li>
</ul>
</nav>

<!-- colonne gauche -->
<section>
<article>

<h1>Titre</h1>
<p>
    Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
</p>
<p>
    Duis aute irure dolor in reprehenderit in voluptate velit esse
    cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
    cupidatat non proident, sunt in culpa qui officia deserunt mollit
    anim id est laborum.
</p>
</article>
</section>

<!-- colonne droite -->
<aside>aside</aside>

<!-- pied de page -->
<footer>footer</footer>

<!-- fin conteneur -->
</div>
```

Les éléments composant la page sont dans un bloc de type **<div>** qui sert de conteneur.

Les différentes balises sémantiques du HTML5 sont utilisées pour créer l'en-tête de la page, le menu, les colonnes de gauche et de droite, et enfin, le pied de page.

CSS

```
<link type="text/css" rel="stylesheet" href="requetesMedia.css" />  
La page HTML appelle un fichier CSS qui va contenir l'ensemble des requêtes de médias selon les conditions définies ci-dessus.
```

Les requêtes de médias

CSS

```
requetesMedia.css  
/* commun */  
@import url("commun.css") only screen and (color);  
  
/* écran large */  
@import url("ecranLarge.css") only screen and (color) and (min-width : 1024px);  
  
/* tablette et netbook */  
@import url("tablette.css") only screen and (color) and (min-width : 480px) and (max-width : 1024px);  
  
/* mobile intelligent */  
@import url("mobileIntelligent.css") only screen and (color) and (max-width : 480px);
```

Un fichier CSS va être utilisé pour les **styles communs** à toutes les largeurs d'écran. Puis, en se basant sur les supports les plus répandus types iPhone et iPad, on suppose qu'un large écran débute lorsque celui-ci dépasse celui de l'iPad en position horizontale, c'est-à-dire à partir de **1 024 pixels minimum**.

Les tablettes et netbooks possèdent une largeur comprise entre les mobiles intelligents et les écrans larges. La largeur définie pour les mobiles intelligents se base sur l'iPhone en position horizontale, c'est-à-dire **480 pixels maximum**.

□ Les styles communs

CSS

```
commun.css  
{*  
padding:0px;  
margin:0px;  
}
```

```
header{  
height:100px;  
background:rgb(200, 200, 200);  
}  
  
nav{  
height:30px;  
background:rgb(180, 180, 180);  
}  
  
nav li{  
list-style-type:none;  
}  
  
nav a{  
width:90px;  
height:30px;  
color:rgb(50, 50, 50);  
font-size:0.8em;  
font-weight:bold;  
line-height:2.2em;  
text-align:center;  
text-decoration:none;  
display:block;  
}  
  
section{  
min-height:350px;  
background:rgb(160, 160, 160);  
}  
  
article{  
margin:20px;  
}  
  
article img{  
float:left;  
margin-right:20px;  
}  
  
article p{  
margin-bottom:10px;  
}
```

```
aside {  
min-height:350px;  
background:rgb(140, 140, 140);  
}  
  
footer{  
height:20px;  
background:rgb(100, 100, 100);  
}
```

Le fichier `commun.css` est le socle indispensable à l'architecture de la page. Il définit principalement les styles liés à son **enrichissement graphique**, comme les couleurs de fond, le corps des textes ou la mise en place du menu horizontal. Concernant les propriétés de **mise en page des blocs**, c'est-à-dire celles qui sont directement affectées par les requêtes de médias, elles ne s'y trouvent pas.

□ Les écrans larges

CSS

```
ecranLarge.css  
#conteneur{  
width:1000px;  
margin:0px auto;  
}
```

```
nav li{  
float:left;  
}
```

```
section{  
width:660px;  
float:left;  
}
```

```
aside {  
width:340px;  
float:left;  
}
```

```
footer{  
clear:both;  
}
```

Pour les écrans larges, les propriétés de mise en page sont :

- la largeur du conteneur est de 1 000 pixels et il est centré dans le navigateur ;
- le menu est défini comme étant horizontal ;
- les deux colonnes ont des largeurs fixes ;
- le pied de page se positionne sous les deux colonnes.

Ces propriétés sont appliquées lorsque la largeur du navigateur fait un minimum de 1 024 pixels, selon notre requête de média.

□ Les tablettes et netbooks

CSS

```
tablette.css
#conteneur{
width:100%;
}
```

```
nav li{
float:left;
}
```

```
section{
width:67%;
float:left;
}
```

```
article img{
width:30%;
height:30%;
}
```

```
aside {
width:33%;
float:left;
}
```

```
footer{
clear:both;
}
```

Pour s'adapter aux nombreuses résolutions existantes, la largeur des blocs devient proportionnelle à la largeur de l'écran pour les tablettes et les netbooks :

- le conteneur s'étend sur l'ensemble de l'écran ;
- le menu reste horizontal ;
- les deux colonnes possèdent des largeurs en pourcentage ;
- l'image se redimensionne proportionnellement à la largeur de l'écran ;
- le pied de page reste sous les deux colonnes.

□ Les mobiles intelligents

CSS

```
mobileIntelligent.css
```

```
#conteneur{  
width:100%;  
}  
  
nav{  
height:auto;  
}  
  
nav li{  
float:none;  
}  
  
nav a{  
width:100%;  
}  
  
section{  
width:100%;  
float:none;  
}  
  
article, article p{  
margin:0px;  
}  
  
article img{  
width:100%;  
height:100%;
```

```
margin:0px;  
float:none;  
}  
  
aside {  
width:100%;  
height:auto;  
float:none;  
}  
  
footer{  
clear:both;  
}
```

Dès que la largeur de la fenêtre du navigateur va atteindre 480 pixels et au-dessous, la mise en page va devenir exclusivement verticale pour s'adapter aux mobiles :

- ▶ le menu devient vertical et sa hauteur n'est plus fixe.
- ▶ les boutons du menu couvrent la largeur de l'écran.
- ▶ les colonnes n'en sont plus, car leur positionnement côté à côté grâce à la propriété *float* est annulé.
- ▶ l'image se redimensionne selon la largeur de la fenêtre du navigateur.
- ▶ le pied de page reprend également un positionnement vertical.



Symboles

@fonte-face 181

154

.Net 8

@import 181

@keyframes 165

@media 182

A-B

accessibilité 13, 117

adresse IP 100

animation 159, 165, 174

API History 119

Backgrounds 146

balise 15

 address 39

 article 28

 aside 30

 audio 38, 50

 canvas 39, 69

 command 41

 datalist 62

 dialog 39

 em 41

 embed 55

 figcaption 37

 figure 37

 footer 27

 header 26

 hgroup 35

 img 53

 input 57

 link 181

 mark 39

 meta 179

 meter 42, 64

 nav 28

 object 53, 54, 55, 56

 progress 42, 63

 section 29

 source 49, 51

 strong 40

 svg 53

 time 41

 video 38, 46

 wbr 41

Border 143

C-D

CMS 8

codecs 45

couleur 82

courbe de Bézier 80, 163

CSS 6

dégradé linéaire 83

dégradé radial 85

déplacement 94, 170

DOCTYPE 18

Dreamweaver 6

F-G-H

Flash 7

formats de polices 154

formulaire 57, 138

géolocalisation 99

HTML 1, 5

I-J

inclinaison 172
 Internet 1
 iPad 179, 185, 188

Mac iOS 12
 matrice 95, 173
 Métadonnées 118
 Microdatas 118
 mise à l'échelle 95, 172
 mobile 9, 177
 moteur 142

PHP 8
 point de référence 173
 positionnement 117

référencement 12
 requêtes de média 177, 180
 Rich Snippets 118
 rotation 93, 171
 sélecteurs CSS3 125
 sémantique 15, 21
 session 108, 110
 shadow 145

tablette 177
 Text-Shadow 149
 transformation 91
 transformation 2D 159, 170
 transition 159, 174
 transparence 151
 type
 color 59
 date 61
 datetime 61
 datetime-local 61
 email 58
 month 62

iPhone 178, 185, 188
 JavaScript 8
 jQuery 69, 70, 88, 99, 101

M-N-O

multi-colonnage 156
 multi-écrans 177
 navigateurs 12
 newsletters 9
 ombre portée 86
 opacité 150

P

pseudo-classes CSS3 125, 128,
 138
 pseudo-éléments CSS3 125,
 140

R-S

Silverlight 7
 son 50
 stockage des données 108
 stockage local 114
 styles 6
 SVG 52, 88
 SWF 54

T-V-W

number 60
 range 60
 search 59
 tel 58
 time 62
 url 58
 week 62
 typographie 153
 vidéo 45
 W3C 2
 Web Storage 109
 Windows 12
 WordPress 8