



02/11/2022

CSS 3

Spécificités : Le Responsive Design

Table des matières

I Web et mobilité	2
II Rappels des principes HTML/CSS pour le RWD	4
1/ Élément HTML et DOM	4
2/ Feuilles de styles CSS	4
3/ Nouveautés CSS3	5
4/ Synthèse : les points-clés	5
III Processus de développement Responsive	6
1/ Mise en page spécifique ou passe-partout ?	6
2/ Structurer d'abord, présenter ensuite	7
3/ Desktop First ou Mobile First ?	7
IV Media queries CSS3	8
1/ Exercice media queries	9
1) Traduire en langage courant les media queries:	9
2) Réponses	10
V Meta viewport HTML	11
1/ Meta HTML viewport	11
2/ Unités de mesure CSS à venir	11
VI Bien structurer une page Web	12
1/ <table> ou <div> ?	12
2/ Exercice : établir une structure HTML5	15
VII Bien utiliser les sélecteurs CSS	17
1/ Notion de poids des sélecteurs CSS	17
2/ Bon usage des attributs HTML class et id	18
VIII Bien utiliser les unités de mesure	19
1/ Rappel sur le système de boîtes CSS	19
2/ La règle d'or pour calculer les dimensions relatives	20
3/ Complément sur la notion de 'em'	23
4/ Grilles fluides	25
IX Images fluides	26
1/ Zoom sur les images	26
2/ Habillage de l'image par le texte	27
X Mise en page utilisant un système de grille CSS	28
1/ Principe de base	28
2/ Exercice : Vers une grille adaptative	28
XI Framework CSS et Préprocesseurs	35
1/ Mise en page avec un Framework CSS	35
2/ Préprocesseurs CSS	35
1) Un préprocesseur CSS, pourquoi faire ?	35
2) Découverte de Sass/Compass	36

I Web et mobilité

Actuellement, les sites Web sont accédés *majoritairement* depuis des terminaux mobiles (smatphones ou tablettes). *Les sites Web conçus et mis en page pour écrans de PC s'affichent mal sur ces terminaux* : nombreux zooms et défilements nécessaires pour afficher lisiblement, ergonomie basée sur boutons et liens graphiques peu adaptés aux commandes tactiles de type *swip* ou *tap*...

De plus, les réseaux de téléphonie mobile historiques étaient *beaucoup plus lents* que les accès ADSL sur réseau de téléphonie fixe et *beaucoup plus sujets à des coupures* (on se croirait revenu au temps des accès par modem 56Kbits !). Il est vrai que maintenant ce n'est plus du tout le cas avec la 4G et la 5G.

Les *tailles et résolutions d'écrans* pouvant accéder à un site Web sont de plus en plus variés et de moins en moins standards (orientations portrait ou paysage, proportions 4/3 ou 16/9, smartphones, tablettes de 7" à 11", tablettes iPad spécifiques, écrans PC de 19" à 27" ou même 30", incertitude sur les terminaux futurs...).



Les *terminaux mobiles disposent de fonctionnalités spécifiques* non prise en compte par les technologies Web actuelles ; même si HTML5/CSS3/JavaScript apportent le support des commandes tactiles, de la géo localisation et du mode hors connexion, l'accès aux contacts, aux fonctions de téléphonie, à l'appareil photo, au gyroscope... restent du domaine des *applications* spécifiques.

Une application spécifique pour mobile doit être développée dans *différents langages et technologies* (iOS, Android, Windows Phone... langages C#, Java... et Framework spécifiques...) et doit être déposée sur des plateformes de téléchargement coûteuses (Google Play, App Store...) ; l'utilisateur doit la *télécharger* et *l'installer* (et la *mettre à jour* régulièrement) sur son terminal pour pouvoir l'utiliser. A l'opposé, un site Web est accessible depuis n'importe quel navigateur et sa mise à jour est instantanée.

On peut considérer que les applications mobiles sont une '*maladie infantile*' de la mobilité, le navigateur Web restant dans son principe le '*client universel*' idéal. Mais, comme les technologies Web intègrent lentement les fonctionnalités mobiles (on n'en serait qu'à la '*toute petite enfance*' !), les développements d'applications spécifiques restent nécessaires encore pour quelques temps...

Le design d'un site Web est communément conçu par un infographiste avec des exigences très précises issues de l'édition imprimée (couleurs, dimensions, placements...); ce design est matérialisé par des fichiers images (Maquettes) que l'équipe de développement se doit de respecter scrupuleusement en utilisant des technologies Web bien moins précises (HTML n'a jamais été conçu pour cela !) et en transposant le design pour différentes tailles d'écrans. C'est le travail quotidien des 'intégrateurs Web', spécialité pointue devenant véritablement un des métiers du Web.

Face à ce constat, plusieurs choix restent possibles :

- Développer **un site Web** pour les PC **et des applications** pour terminaux mobiles incluant des fonctionnalités supplémentaires → développement et maintenance très lourds ; affichage et ergonomie parfaits ; **service maximum**.
- Développer **plusieurs versions du site Web**, chacune étant optimisée pour un type de périphérique (solution historique) → maintenance très lourde ; affichage et ergonomie corrects ; **adapté aux sites ne nécessitant pas d'accès aux fonctions spécifiques des terminaux mobiles**.
- Développer **un seul site Web** exploitant le maximum des fonctionnalités Web du moment et **s'adaptant automatiquement au périphérique** de l'utilisateur → développement encore délicat, maintenance aisée, affichage et ergonomie corrects ; **adapté aux sites ne nécessitant pas d'accès aux fonctions spécifiques des terminaux mobiles**.

Cette dernière solution relève de ce qu'on appelle le **Responsive Web Design** ou *conception de sites Web réactifs* ; elle est basée sur une bonne utilisation de HTML5 et CSS3 en appliquant 3 principes fondamentaux : **grille fluide** (la mise en page s'adapte à la taille de l'écran), **images fluides** (les tailles des images s'adaptent aussi) et technique CSS des **media queries** (qui permet d'identifier finement et dynamiquement le type de périphérique utilisé pour choisir les styles à appliquer aux éléments HTML de la page Web).

Le but de cette **adaptation automatique des pages Web aux caractéristiques physiques du périphérique** qui les restitue est de conserver le maximum d'information, de fonctionnalités des pages, en utilisant au minimum le masquage d'éléments (display : none) sur petit écran mais **en faisant varier la mise en page et la taille des éléments** à afficher, sans pour autant nécessiter des défilements et zooms incessants par l'utilisateur. Pas si facile à dire ; encore moins facile à réaliser... Des outils peuvent maintenant aider (Framework CSS comme *Bootstrap* ou *Foundation*) au prix d'une augmentation considérable du volume des pages Web.

II Rappels des principes HTML/CSS pour le RWD

1/ Élément HTML et DOM

HTML mélange historiquement structure du document (<body>, <h2>...) et présentation du contenu (, ...). CSS a été conçu pour séparer structure et présentation des pages Web :

- **HTML définit uniquement la structure de la page,**
- **Les styles CSS s'occupent du rendu graphique.**

Ainsi, un code HTML bien écrit ne doit plus contenir ni balises, ni attributs HTML de présentation.

Dans le code HTML, **toute balise définit un élément de la page** ; une balise double comme <div>... </div> définit un élément possédant un contenu (tout ce qui est entre les balises ouvrante et fermante) ; une balise simple comme
 ou <input type = "text" name=... /> définit un élément vide.

Chaque élément HTML est vu par CSS comme un 'objet' ; ses **attributs HTML** (type, name, width...) sont alors vus comme les **propriétés de l'objet** ; les valeurs de ces propriétés sont définies initialement par le code HTML mais peuvent à tout moment être dynamiquement modifiées, soit par JavaScript, soit par CSS.

Quand elle n'est pas précisée dans le code, **une propriété HTML a toujours une valeur par défaut**. Les valeurs par défaut peuvent varier d'un navigateur à l'autre mais elles tendent à se normaliser. Tout navigateur ajoute aux standards courants ses propres extensions, aux niveaux HTML, CSS, JavaScript... Rappelons qu'**une propriété HTML ou CSS inconnue du navigateur sera ignorée sans message d'erreur**.

Tout attribut HTML a son équivalent en propriété CSS, et **CSS offre une multitude de propriétés en supplément** permettant d'exploser les capacités de rendu graphique de HTML (animations, bordures arrondies, ombrages, positionnements...).

Une page Web est une structure arborescente : un élément est contenu dans un élément , tout élément de la page est inclus dans l'élément <body>.

Pour représenter tout cela, on parle couramment du '**Document Object Model**' ou 'DOM', introduit par le langage JavaScript, qui définit une page Web comme un **arbre d'objets reliés les uns aux autres**. La technique fondamentale des sélecteurs CSS s'appuie sur le DOM.

2/ Feuilles de styles CSS

Si les styles CSS peuvent encore être définis, pour des raisons historiques, localement sur un élément (attribut HTML style=...) ou en entête de page Web (balise HTML <style>...</style>), **le bon usage de CSS impose l'écriture de feuilles de styles externes**, séparées du code HTML, et **réutilisables** par toutes les pages Web concernées.

Il est alors nécessaire, avant tout codage CSS, d'effectuer une analyse du problème afin de définir un découpage de l'ensemble des règles nécessaires au sein de plusieurs fichiers CSS, chacun regroupant des règles de styles formant un ensemble ou sous-ensemble cohérent.

Au-delà du découpage habituel aboutissant à une feuille pour tout périphérique, une autre pour l'écran et une dernière pour l'impression, **l'approche Responsive Web Design nécessite d'identifier précisément les différents périphériques-types ciblés afin de définir les sous-ensembles, de les regrouper en fichiers CSS nécessaires et de minimiser ainsi le volume de code CSS** à télécharger, surtout pour les mobiles plus lents et moins puissants. L'approche récente 'Mobile First' (voir plus loin) va bien dans ce sens.

3/ Nouveautés CSS3

Pour terminer cette mise au point sur les bases HTML/CSS, voici un bref rappel des principales nouveautés CSS3. Notons que CSS3 est défini par un ensemble complexe de standards (des *chapitres*) qui sont spécifiés progressivement par les équipes du W3C ; ainsi certaines fonctionnalités sont déjà bien stabilisées alors que d'autres ne sont qu'à peine ébauchées. Et on parle déjà des 'sélecteurs CSS4'...

Pendant les nécessaires périodes de transition, chaque moteur de navigateur définit et implémente sa propre vision de la fonctionnalité avant que le standard ne soit stabilisé ; ainsi pour définir une bordure arrondie, la propriété CSS standard est `border-radius`, mais le monde FireFox a de plus inventé `-moz-border-radius`, celui de Chrome, `-webkit-border-radius` et le monde Apple, `-o-border-radius` quand Internet Explorer proposait `-ms-border-radius`...

Au final, pour raison de compatibilité avec d'anciens navigateurs, on écrit bien souvent 5 règles de styles identiques pour s'assurer que chacun y trouvera une règle applicable ! Dans ce document, on se limitera aux propriétés standards pour ne pas compliquer inutilement les apprentissages.

4/ Synthèse : les points-clés

Dans cette partie 'Web et Mobilité', il faut bien retenir :

- **HTML5/CSS3 ne remplacent pas aujourd'hui toutes les applications mobiles** car de nombreuses fonctionnalités des tablettes et smartphones ne sont pas disponibles à travers la programmation de pages web.
- **HTML est fait pour structurer** l'information de la page Web et **CSS pour définir la présentation**.
- Toute balise **HTML** définit un **élément du document**, vu comme un '**objet**' par **CSS**.
- Tous les attributs HTML ont leur équivalent en CSS (et bien d'autres) ; **tout ce qui est défini par HTML peut se modifier/contredire par CSS**.
- En général, les propriétés CSS d'un **élément enfant** sont **héritées** de celles de son **élément parent** (son conteneur) ; on peut encore **interdire ou forcer** cet héritage, propriété par propriété.
- Les propriétés CSS **display et float**, combinées avec le principe d'élément **block/inline** permettent de **contrôler la mise en page** (ne pas utiliser de tableaux HTML pour la mise en page).
- Les **sélecteurs CSS3** permettent de **cibler** très précisément **un élément ou un ensemble d'éléments** du document, pour **fixer une règle de styles**, sans devoir intervenir dans le code HTML.

Le bon usage des feuilles de styles CSS passe par le regroupement, dans des fichiers externes à la page Web, de jeux de styles réutilisables.

III Processus de développement Responsive

1/ Mise en page spécifique ou passe-partout ?

Aujourd'hui, il est fréquent que le design graphique soit assuré par un infographiste qui livre les *maquettes* des différentes pages-types du site sous forme de *fichiers graphiques* (PhotoShop ou Figma).

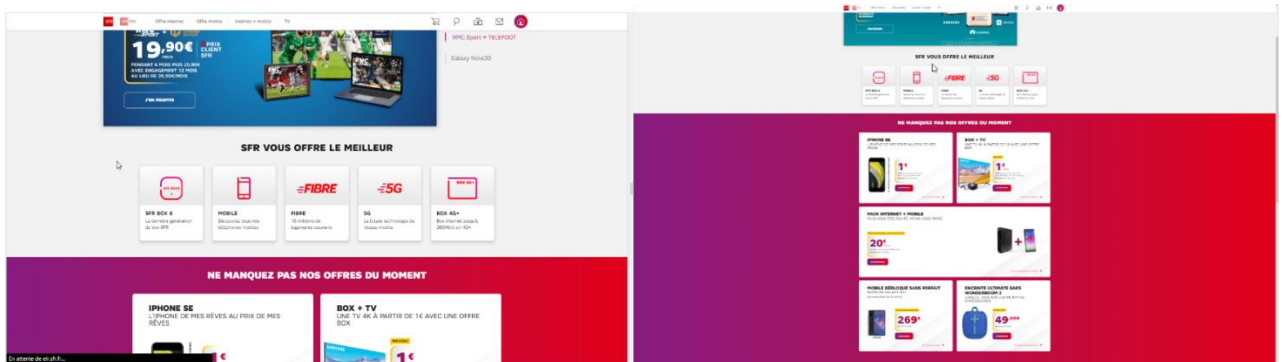
Cette pratique est issue du monde de l'imprimerie pour lequel un fichier PhotoShop peut être traité directement par une photocomposeuse afin de graver les plaques d'impression nécessaires aux machines d'imprimerie.

Mais dans le monde Web, il reste encore compliqué de traduire une maquette PhotoShop en code HTML/CSS, et encore moins en code 'responsive' ! Alors les « WebDesigner », après avoir défini la structure HTML des pages, sont amenés à analyser ces maquettes pour y mesurer les dimensions précises des éléments (en cm, en pixels...) et identifier précisément les couleurs afin de définir manuellement les règles de styles CSS permettant de reproduire le plus fidèlement possible les maquettes fournies.

On peut se rendre compte de la problématique en 'feuilleter' la vidéo Grafikart « Intégrer une maquette » (<http://www.grafikart.fr/tutoriels/html-css/integration-front-end-497> - durée 2h45mn !).

Mais cette pratique se heurte aujourd'hui à la *nécessaire adaptation de la mise en page par rapport au périphérique qui restitue la page Web*. Par exemple, si la maquette a été définie pour une largeur de 1050 pixels, comment l'adapter de manière esthétique et optimale à un écran 16/9 de 24 pouces qui peut afficher 1680 pixels ?

On a coutume dans ce cas de centrer à l'écran la partie 'utile' de la page en faisant étendre la couleur ou l'image de fond sur toute la largeur ; ce n'est pas une très bonne gestion de l'espace disponible...



Quant aux utilisateurs de tablettes et smartphones ils devront user et abuser des ascenseurs et de la loupe pour consulter les informations... L'idéal est de réaliser de véritables effets de zoom sur une même plage de dimensions et de varier la mise en page selon le type de périphériques ; c'est tout l'objet du Responsive Web Design !

2/ Structurer d'abord, présenter ensuite

Typiquement, le développement d'un site *responsif* **commence** par la définition de la **structure HTML** de chaque page-type, *hors présentation* :

- En utilisant au mieux les **éléments structurants HTML5** (<header>, <nav>... et aussi , ...)
- En évitant les **imbrications trop nombreuses d'éléments <div>**
- **En évitant toute mise en page par des tableaux HTML** (<table> utilisable uniquement pour de véritables tableaux de données)
- En ajoutant éventuellement des éléments structurants qui faciliteront l'identification des portions à styler grâce aux sélecteurs CSS3 plutôt qu'aux attributs HTML id
- En **limitant l'usage** des attributs **HTML class et id** au profit des **sélecteurs CSS3**, conseil qui est bafoué par les Framework.

A partir de la maquette graphique, le développement se poursuit :

- En définissant progressivement les **sélecteurs et styles CSS** à appliquer pour obtenir le rendu le plus fidèle à la maquette graphique (codes couleurs, dimensions et placements au pixel près)
- Puis en définissant **dimensions et placements de manière relative** (en em, en rem, en %) de manière à assurer un rendu adaptatif dans la plage de dimensions correspondant à cette maquette
- En ajoutant ensuite progressivement les jeux de sélecteurs et styles assurant la mise en page et le rendu graphique pour des média ou plages de dimensions différents grâce aux '**media queries**' CSS3.

Avec ce procédé, la mise en page reste fluide et harmonieuse à l'intérieur d'une même plage de dimensions (zoom) mais elle subit de brusques changements quand on dépasse les points de rupture de la plage de dimensions (le développeur met cela en évidence en faisant varier à la souris la taille du navigateur pendant les tests mais il faut garder à l'esprit que l'utilisateur, lui, ne dispose que d'un périphérique particulier à un moment donné).

A chaque étape, des phases de **tests et validations** sont bien sûr nécessaires.

Structurer en HTML d'abord, présenter par CSS ensuite

3/ Desktop First ou Mobile First ?

A l'origine, on avait coutume de définir en premier la mise en page pour les PC de bureau (démarche *Desktop First*) mais on s'est vite rendu compte que cela entraîne une multitude d'annulations/rectifications de règles de styles pour les écrans plus petits (tablettes et smartphones) ; en conséquence, cela provoque une augmentation du volume des pages pour les mobiles, alors que ces périphériques étaient sensiblement moins puissants.

La tendance actuelle est donc d'adopter la démarche inverse, dite *Mobile First*, qui consiste à définir en premier les styles nécessaires à la mise en page pour mobiles, puis d'ajouter des règles de styles et *media queries* nécessaires pour tablettes, écrans larges, voire très larges.

Il est temps de mettre du concret dans tout cela et d'approprier quelques techniques HTML et CSS.

IV Media queries CSS3

Un '**media query**' CSS permet d'identifier un type précis de périphérique de restitution d'une page Web en précisant :

- Le **type** de media (print, screen, aural, all...)
- Une ou plusieurs expressions qui définissent des **caractéristiques physiques** du media (width, orientation, resolution, color...)

Voir la documentation de référence sur les media queries :

- <http://www.w3.org/TR/css3-mediaqueries/>
- <http://www.w3.org/TR/CSS21/media.html>
- https://developer.mozilla.org/fr/docs/Web/CSS/Media_Queries/Using_media_queries

Une expression de *media query* peut être utilisée aussi bien pour :

- Définir, dans le code CSS, des groupes de règles des styles CSS à utiliser selon le périphérique de restitution → directive CSS @media... {...}
- Définir des feuilles de styles CSS à télécharger et à utiliser selon le périphérique de restitution → balise HTML <link rel="stylesheet" media="..." href="..." />

CSS3 définit une syntaxe très puissante permettant de cibler aisément :

- Une plage de largeur d'écran selon le nombre de pixels affichables (smartphone, tablette, PC ordinaire, grand écran, téléviseur connecté...);
- Un type de périphérique particulier (écran, imprimante, clavier braille et même lunette 3D);
- Une orientation d'écran (le jeu de styles s'applique dynamiquement lorsque l'utilisateur pivote son périphérique mobile);
- Le support ou non de la couleur...
-

L'équipe de développement se doit donc **d'identifier les périphériques-types** courants **ciblés par le site Web**; ultérieurement, lors de l'apparition d'un nouveau type de périphérique (lunettes 3D, montre connectée...), il '*suffira*' de développer un nouveau jeu de styles à appliquer.

Les media queries peuvent être utilisées de deux manières différentes.

Soit en le définissant dans le link CSS pour appeler un fichier en fonction du contexte défini.

Fichier HTML

```
<link rel="stylesheet" href="mobile.css" media="screen and (max-width :640px)">
```

Soit en définissant les règles CSS pour le media query concernant dans un fichier CSS.

Fichier HTML

```
<link rel="stylesheet" type="text/css" href="style_exercice1_CSS3.css">
```

Fichier CSS

```
@media screen and (min-width :200px) and (max-width :639px) {  
  
}
```

1/ Exercice media queries

1) Traduire en langage courant les media queries:

L'objectif est de traduire les media queries ci-dessous en langage courant.

<code><link rel="stylesheet" media="screen and (max-width :640px)" href="xx.css"></code>
<code>@media screen and (min-width :200px) and (max-width :639px) and (orientation:landscape) {...}</code>
<code>@media handheld and (orientation:portrait)</code>
<code><link rel="stylesheet" media="print and (min-width :5in)" href="xx.css" type="text/css" /></code>
<code>@media tv, (device-aspect-ratio :16/9), (device-aspect-ratio :16/10) {...}</code>
<code><link rel="stylesheet" media="only screen and (color)" href="yy.css" type="text/css" /></code>

Traduire en media queries les définitions de périphériques exprimées ci-dessous en langage courant :

Feuille de styles externe pour tablettes disposant d'un écran affichant entre 768 et 991 pixels en largeur en orientation portrait
Jeu de styles CSS pour écrans extra-larges (affichant au moins 1200 pixels)
Jeu de styles seulement pour l'impression si la résolution est d'au moins 300 points par pouce

Réponses page suivante.

2) Réponses

Traduire en langage courant les media queries ci-dessous :

<code><link rel="stylesheet" media="screen and (max-width :640px)" href="xx.css"></code> Feuille de styles liée pour des écrans affichant au plus 640 pixels en largeur
<code>@media screen and (min-width :200px) and (max-width :639px) and (orientation:landscape) {...}</code> Jeu de styles pour écrans en orientation paysage affichant une résolution entre 200 et 639 pixels en largeur
<code>@media handheld and (orientation:portrait)</code> Jeu de styles pour périphériques mobiles en orientation portrait
<code><link rel="stylesheet" media="print and (min-width :5in)" href="xx.css"></code> Feuille de styles liée pour des imprimantes disposant d'au moins 5 pouces en largeur
<code>@media tv, (device-aspect-ratio :16/9), (device-aspect-ratio :16/10) {...}</code> Jeu de styles pour téléviseurs et périphériques affichant en format 16/9 ou 16/10
<code><link rel="stylesheet" media="only screen and (color)" href="yy.css"></code> Feuille de styles liée uniquement pour des écrans affichant en couleurs

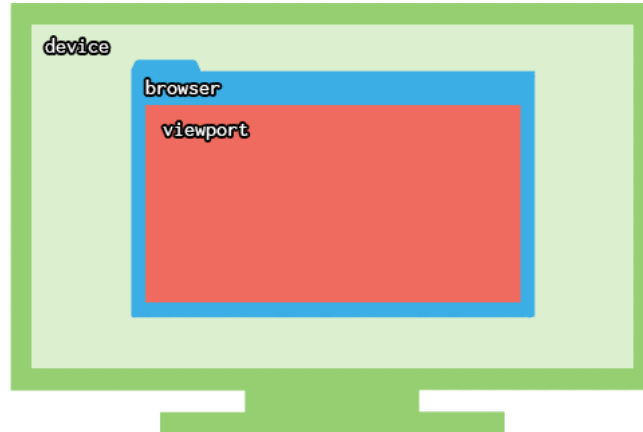
Traduire en media queries les définitions de périphériques exprimées ci-dessous en langage courant :

Feuille de styles externe pour tablettes disposant d'un écran affichant entre 768 et 991 pixels en largeur en orientation portrait <code><link rel="stylesheet" media="screen and (min-width :768px) and (max-width :991px) and (orientation = portrait)" href="xx.css"></code> <code><link rel="stylesheet" media="handheld and (min-width :768px) and (max-width :991px) and (orientation = portrait)" href="xx.css"></code> <i>Cette deuxième solution est plus risquée car le média handheld reste mal déterminé...</i>
Jeu de styles CSS pour écrans extra-larges (affichant au moins 1200 pixels) <code>@media screen and (min-width :1200px) {...}</code>
Jeu de styles seulement pour l'impression si la résolution est d'au moins 300 points par pouce <code>@media only print and (min-resolution: 300dpi){...}</code>

V Meta viewport HTML

1/ Meta HTML viewport

Le Viewport désigne schématiquement la surface de la fenêtre du navigateur. Sur terminaux mobiles, les pages web sont *dézoomées* par défaut pour entrer dans la surface du terminal (une composition pour écran de PC est bien souvent illisible sur un mobile du fait de ce zoom réduit). La balise HTML <meta> peut maintenant servir à contrôler les dimensions d'affichage initial et le niveau de zoom que peut effectuer l'utilisateur.



```
<meta name="viewport" content="width=device-width">
```

Assure un affichage au maximum de la largeur du navigateur (intéressant pour un affichage sur mobile ou tablette).

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

Dans ce cas, aucun navigateur ne zoome plus la page, ce qui est bien pratique pendant la phase de mise au point des styles CSS.

```
<meta name="viewport" content="width=device-width, minimum-scale=0.5, maximum-scale = 3.0, user-scalable=yes">
```

Autorise l'utilisateur à zoomer sur la page entre demi-taille (0.5) et trois fois la taille (3.0) d'affichage du périphérique.

Pour vous aider à mieux comprendre le viewport :

<https://www.alsacreations.com/article/lire/1490-Comprendre-le-Viewport-dans-le-Web-mobile.html>

2/ Unités de mesure CSS à venir

A l'avenir, de nouvelles unités de mesure CSS (vh, vw) pourront être utilisées pour définir des dimensions proportionnellement aux caractéristiques du *viewport*.

1 vh = 1/100 de la hauteur du viewport

1 vw = 1/100 de la largeur du viewport

*Attention si votre PC a une résolution d'écran de 1920*1080, la hauteur du viewport sera égale à : 1080 – la hauteur que prendra le menu du navigateur (barre d'adresse + barre de favoris + barre d'onglets etc..)*

VI Bien structurer une page Web

1/ **<table> ou <div> ?**

Les **tableaux HTML** (éléments `<table>`, `<tbody>`, `<tr>`, `<td>`...) ont longtemps été utilisés aussi bien pour **présenter des tableaux de données** que pour **assurer une mise en page** relativement adaptative des pages Web.

Ce dernier usage est totalement obsolète depuis HTML4 (bien que très efficace) et il est recommandé de structurer les pages Web à l'aide des éléments HTML `<div>`. Mais comme ces éléments ne disposent d'aucun rendu par défaut (à part la présentation en *block*), il est nécessaire d'assurer le rendu visuel à l'aide de styles CSS, assez lourds, mais qui autorisent des présentations variées pour une même structuration HTML (placement côte à côte ou dessus-dessous selon la largeur d'écran).

Face à la '*crise de divite augüe*' qui a envahi les pages Web modernes, HTML5 apporte de nouveaux éléments structurants (qui ne disposent eux aussi d'aucune présentation par défaut, tout restant à définir dans les styles CSS).

Ces éléments structurants permettent de **donner du sens aux différentes parties de la page** ; à terme, ils seront de plus pris en compte par les moteurs de recherche pour affiner la pertinence des résultats de recherches. Tout développement Web actuel se doit d'utiliser ces nouveaux éléments HTML5.

Dans les pages suivantes nous allons étudier les avantages de la structuration HTML d'un document.

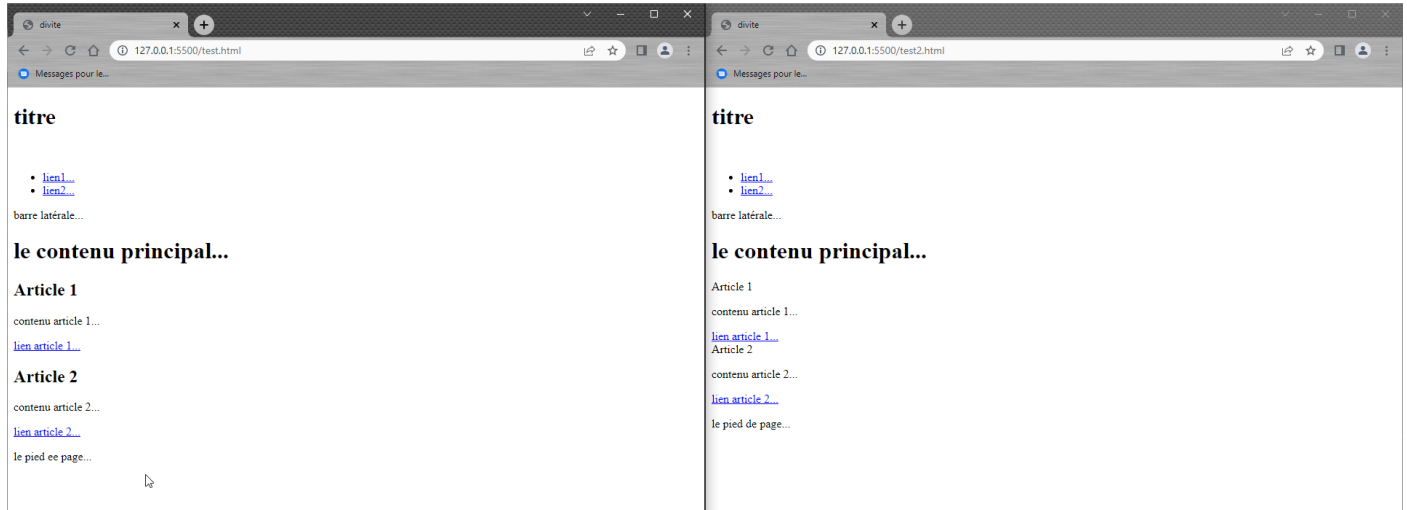
Exemple de structuration classique à base de <div> :

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>divite</title>
</head>
<body>
  <div id="wrapper">
    <!-- entete et navigation -->
    <div id="header">
      <div id="logo">
        <h1>titre</h1>
        <img src="" />
      </div><!-- logo -->
      <div id="navigation">
        <ul>
          <li><a href="#">lien1...</a></li>
          <li><a href="#">lien2...</a></li>
        </ul>
      </div> <!-- navigation-->
    </div><!-- entete et navigation -->
    <div id="sidebar">
      <p> barre latérale...</p>
    </div> <!-- sidebar -->
    <div id="main">
      <h1>le contenu principal...</h1>
      <h2>Article 1</h2>
      <p>contenu article 1...</p>
      <a href="#">lien article 1...</a>
      <h2>Article 2</h2>
      <p>contenu article 2...</p>
      <a href="#">lien article 2...</a>
    </div><!-- main -->
    <div id="footer">
      <p>le pied ee page...</p>
    </div><!-- footer -->
  </div><!-- wrapper-->
</body>
</html>
```


La même page structurée différemment avec les balises HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>divite</title>
</head>
<body>
  <div id="wrapper">
    <!-- entete et navigation -->
    <header>
      <div id="logo">
        <h1>titre</h1>
        <img src="" />
      </div><!-- logo -->
      <nav>
        <ul>
          <li><a href="#">lien1...</a></li>
          <li><a href="#">lien2...</a></li>
        </ul>
      </nav>
    </header><!-- entete et navigation -->
    <aside>
      <p> barre latérale...</p>
    </aside>
    <div id="main">
      <h1>le contenu principal...</h1>
      <article>
        <header>Article 1</header>
        <p>contenu article 1...</p>
        <nav>
          <a href="#">lien article 1...</a>
        </nav>
      </article>
      <article>
        <header>Article 2</header>
        <p>contenu article 2...</p>
        <nav>
          <a href="#">lien article 2...</a>
        </nav>
      </article>
    </div><!-- main -->
    <footer>
      <p>le pied de page...</p>
    </footer>
  </div><!-- wrapper-->
</body>
</html>
```

Rendu dans les 2 cas

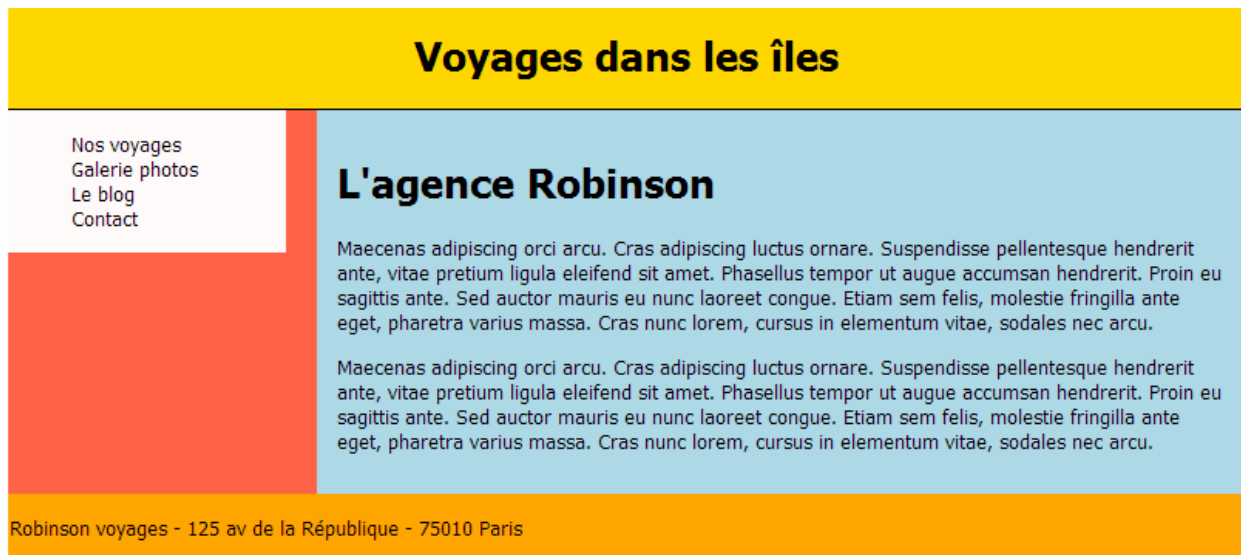


La structure est parfois plus lourde (voir les articles) mais plus claire (moins de commentaires HTML) et le rendu par défaut sera quasiment le même, mais nous avons **ajouté du sens** aux différentes parties de la page, et les moteurs de recherche utiliseront peu à peu ces éléments structurants pour apporter encore plus de pertinence dans les résultats de recherches.

De plus, la variété des éléments HTML facilite l'usage des sélecteurs CSS. Reste à présenter le tout.... Grâce à CSS.

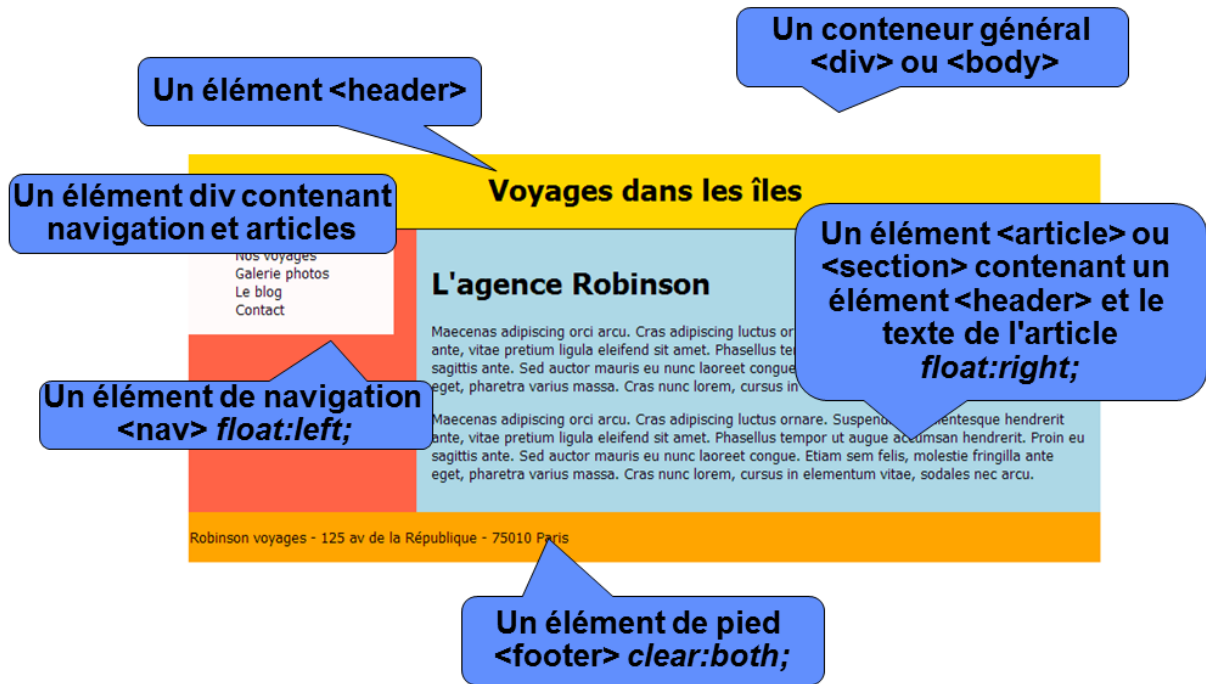
2/ Exercice : établir une structure HTML5

Etablir une structuration HTML5 applicable à la page Web ci-dessous, présentée ainsi sur écran PC et smartphone (sans s'occuper du code CSS pour l'instant).



Eléments de solution page suivante.

Éléments de solution pour structurer la page 'Voyages dans les îles' :



Soit l'exemple de structure HTML5 :

```
<body>
  <!-- conteneur principal -->
  <header>Voyages dans les îles</header><!-- titre général (jaune) -->
  <div>
    <!-- colonne gauche (rouge)-->
    <nav>
      <!-- bloc navigation -->
      <ul>
        <!-- menu -->
        <li>Nos voyages</li>
        <li>Galerie photos</li>
        ...
      </ul>
    </nav>
    <section>
      <!-- ou <main> ; colonne principale (bleue) -->
      <article>
        <!-- un article -->
        <header>L'agence Robinson</header><!-- titre -->
        <div>
          <!-- contenu article -->
          <p>...</p>
          <p>...</p>
        </div>
      </article>
    </section>
  </div>
  <footer>Robinson voyages...</footer><!-- pied de page -->
</body>
```

VII Bien utiliser les sélecteurs CSS

1/ Notion de poids des sélecteurs CSS

Plus un sélecteur CSS est précis, mieux il s'applique ; c'est une conséquence directe du principe CSS d'héritage en cascade.

Ainsi, on peut définir un style pour tous les éléments (*), le préciser ou rectifier pour un élément-type particulier (h1) et le préciser/rectifier encore pour une variante de ce même élément (h1.rouge) ou encore pour un élément identifié (#menu) :

```
* { color : black ; }
h1 {color : blue ; }
h1.rouge {color : red ; }
#menu {color : green ; }
```

Le sélecteur h1.rouge est **plus précis** que h1, lui-même plus précis que * ; h1.rouge a donc un **poids CSS plus fort** que h1 qui a lui-même un poids plus fort que *.

Pour concrétiser cette règle, CSS définit des poids-type pour chaque type de sélecteur et un mode de calcul, respectés par tous les navigateurs.



Poids-types des sélecteurs :

Sélecteur CSS	poids				exemple
*				0	* {...}
élément				1	h1 {...}
.class			1	0	.gras {...}
:pseudo-classe					ul:first-child{...}
#id		1	0	0	#titre {...}
!important	1	0	0	0	...{width:10em !important}
style inline (en code HTML)	1	0	0	0	<... style="..." ...>

Exemples de calculs de poids :

```
#header .logo img {...}      ➔ 100+10+1 = 111
div#header a.logo img {...}  ➔ 1+100-1+10+1 = 113
.logo > img {...}             ➔ 10+1=11
.logo img {...}               ➔ 10+1=11
#page div.widget p a.special{...} ➔ 100+1+10+1+1+10 = 123
```

NB : En cas de conflit entre règles de styles, le sélecteur de poids le plus fort est pris en compte

2/ Bon usage des attributs HTML class et id

Les sélecteurs CSS sont donc très puissants et bien souvent plusieurs syntaxes différentes permettront de sélectionner les mêmes éléments d'une page HTML.

Voici quelques règles simples à respecter pour un bon usage des sélecteurs CSS :

- **Eviter de sélectionner sur les attributs HTML id** (styles #xxx{...}) et **class** (styles .xxx{...})
- **Eviter les sélecteurs trop longs** au profit de sélecteurs plus courts, plus pertinents et au poids plus léger, et ainsi faciliter la maintenance évolutive des pages

NB : ces deux consignes visent à limiter le poids des sélecteurs CSS et sont parfois contradictoires...

NB : l'utilisation de Framework CSS entraîne un usage massif des attributs class (voir plus loin) ...

Dans l'exemple façon 'divite' donné ci-dessus, on peut sélectionner les titres des articles ainsi :

#wrapper #main h2{...} → chemin absolu ; poids 201

#main h2 {...} → plus court et non ambigu ; poids 101

Mais ces deux syntaxes sélectionnent tous les titres <h2> du contenu principal et il pourrait bien y en avoir d'autres ailleurs ; en cas d'ambiguïté, il sera alors nécessaire d'ajouter de la profondeur dans le sélecteur ou d'ajouter des attributs id ou class dans le code HTML.

Or, dans une équipe de développement Web, certaines personnes produisent du contenu (auteurs), d'autres développent les feuilles de styles (infographistes et intégrateurs Web) et d'autres encore développent les pages HTML statiques et dynamiques (développeurs Web).

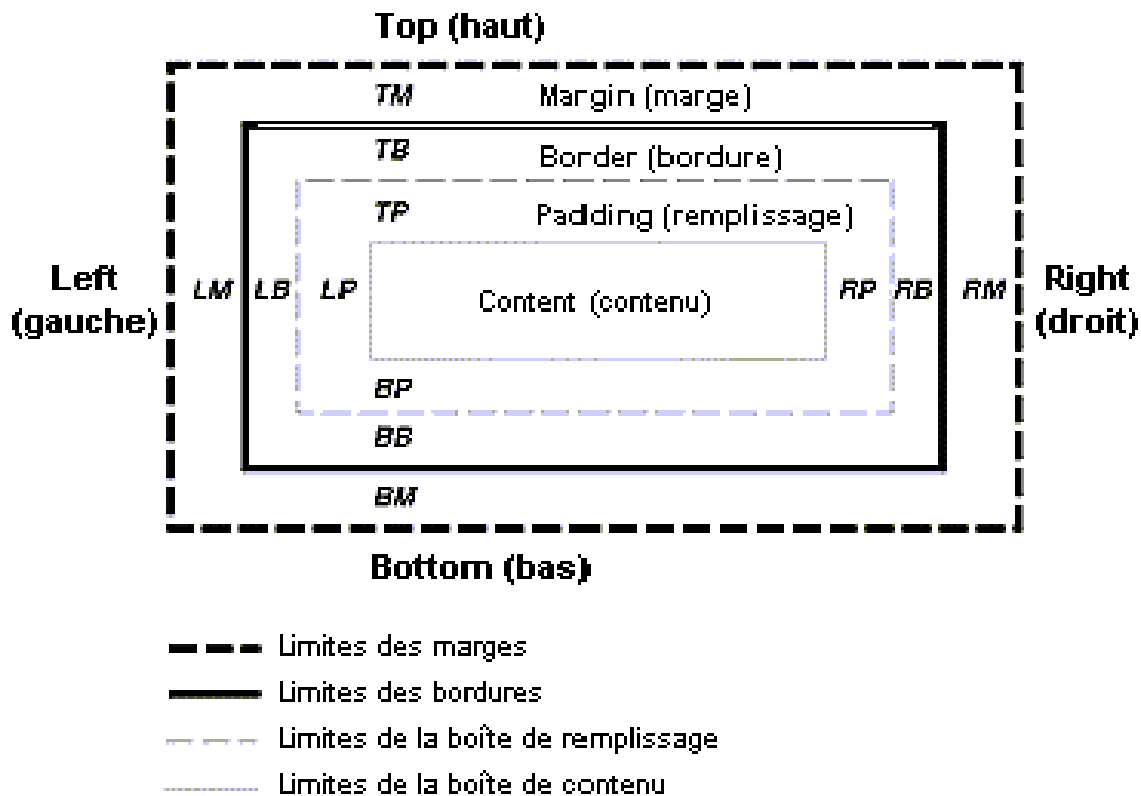
Dans ce cas de travail en équipe, il est souvent préférable que la présentation par CSS se calque sur la page avec un minimum d'intervention sur le code HTML (usage massif des sélecteurs CSS au détriment des attributs id et class HTML).

Selon le même exemple de page structurée avec HTML5, on accède aux titres des articles ainsi :
article header {...} → aucune ambiguïté et du sens à cette sélection, poids 2

VIII Bien utiliser les unités de mesure

1/ Rappel sur le système de boîtes CSS

CSS est basé sur un système de boîte ; chaque élément HTML de type *bloc* (*display :block* ou *display :inline-block*) peut être considéré comme une boîte disposant de ses propres marges internes (*padding*), externes (*margin*) et bordures (*border*).



Les dimensions de l'élément ne se réduisent donc pas à celles de son contenu ; il faut y ajouter marges et bordures :

Largeur d'affichage d'un élément =

margin-left + border-left + padding-left + contenu + padding-right + border-right + margin-right

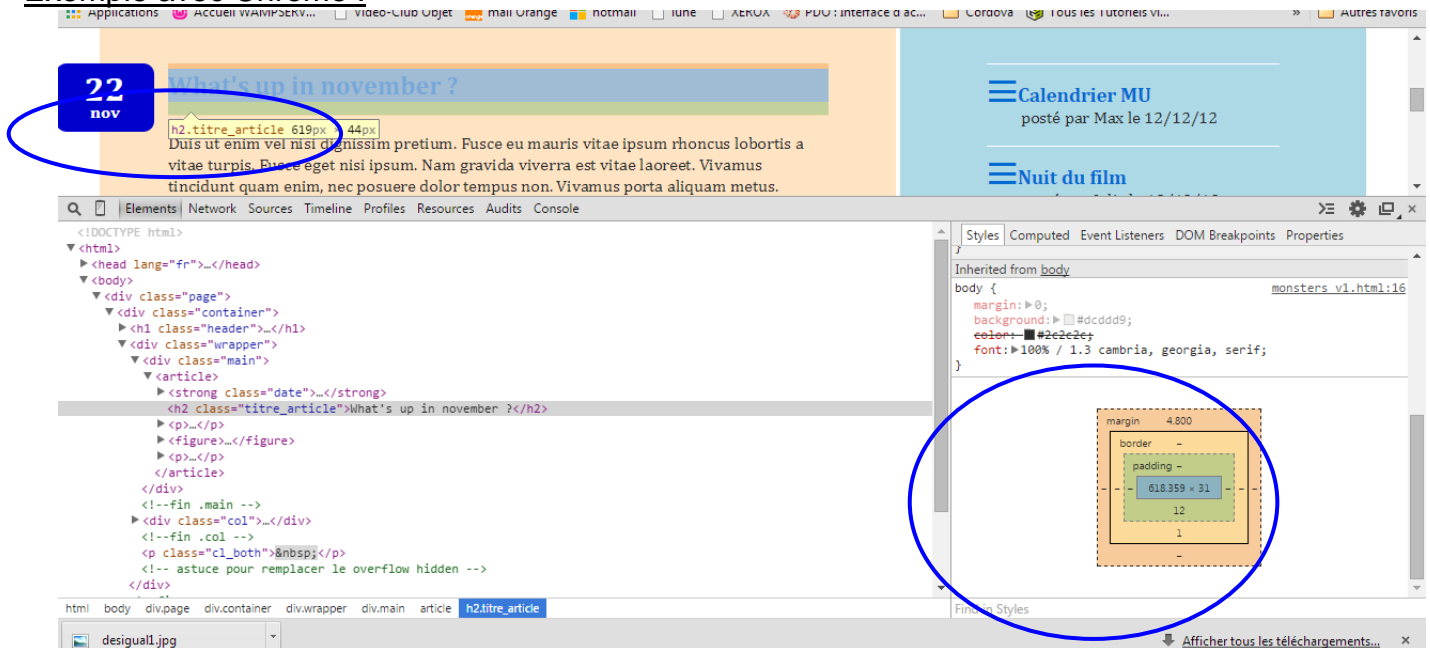
La propriété CSS **box-sizing** admet 2 valeurs :

- **content-box** (par défaut) : la largeur donnée par le développeur (*width*) doit tenir compte des padding et margin.
- **border-box** : la largeur donnée (*width*) **inclut** les padding et margin.

Cette option **box-sizing :border-box** est nécessaire pour obtenir des calculs précis en RWD ; elle doit être définie en début de feuille de styles.

Enfin, reprenez que tous les navigateurs actuels incluent des outils de mise au point (débogueur) qui affichent aisément les dimensions d'un élément sélectionné dans la page :

Exemple avec Chrome :



2/ La règle d'or pour calculer les dimensions relatives

Le Responsive Design est basé sur 2 principes :

- **Mises en page variables selon les périphériques et des plages de largeur, identifiées par media queries** (par exemple : écrans paysage de 1050 à 1680 pixels en largeur, écran portrait de moins de 200 pixels de largeur...)
- **Adaptation automatique à la résolution du périphérique à l'intérieur d'une même plage** (occupation de toute la largeur disponible par **effets de zoom** et non par augmentation de marges).

Les media queries permettent déjà de cibler des types de périphériques (largeur écran de 1050px à 1680px par exemple) afin de déterminer des mises en page différentes.

Ainsi certaines informations moins indispensables pourront ne pas être affichées (display :none) et la mise en page verticale sur une seule colonne (placement dessus-dessous plutôt que côte à côte) sera privilégiée sur un matériel mobile.

De plus, on souhaite que *dans une même plage de media query, la mise en page reste identique et proportionnelle à la définition de l'écran* (effet de zoom) afin d'occuper tout l'espace disponible ; les dimensions des éléments, des blocs, voire des images, les valeurs de marges et d'espacements doivent donc rester **proportionnels**.

C'est là qu'intervient la règle d'or :

Dimension = cible / contexte

Exprimer les dimensions en unités relatives (% , em ou rem)

Exemple : sur une maquette composée sur 1050 pixels de large, un cadre mesure 850 pixels.

On définira donc que la page occupera 100% de la largeur disponible (width de l'élément <body> ou <html>) et que l'élément <div> correspondant au cadre aura, lui :

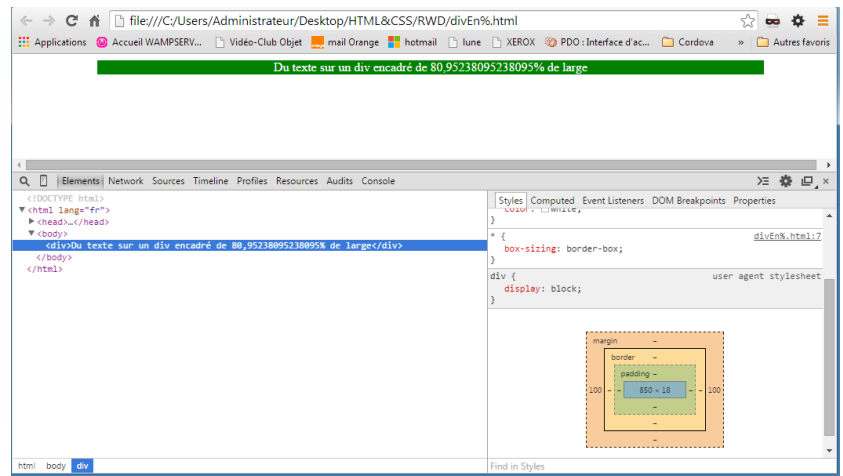
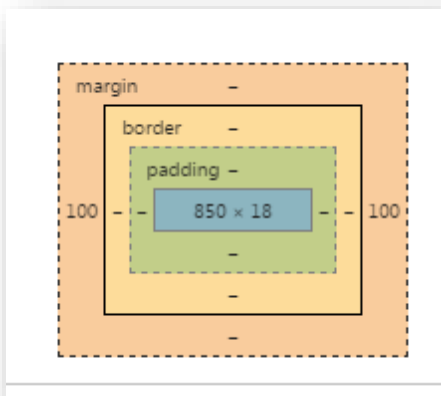
Une largeur de $850/1050 = 0,8095238095238095$ ou 80.95238095238095% soit le règle CSS :

{width : 80.95238095238095%;} /* 850 / 1050 */

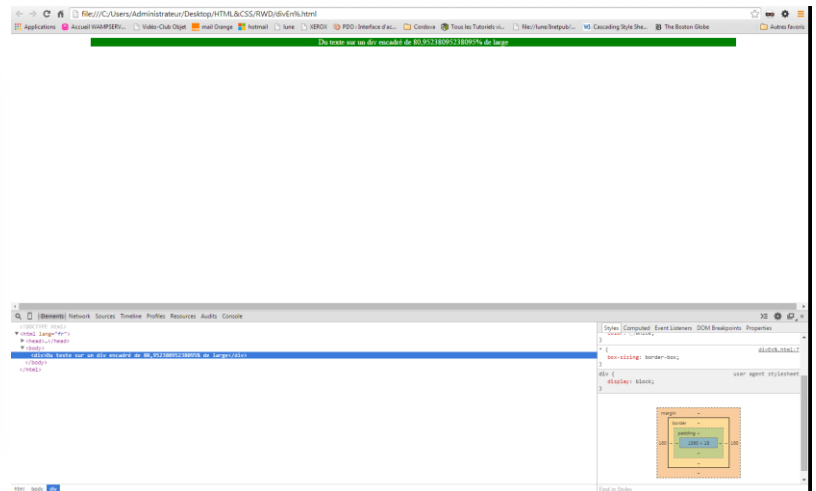
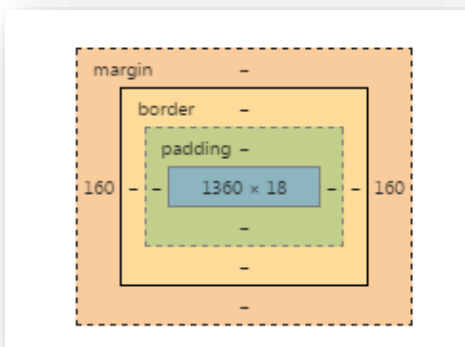
Ainsi sur l'écran 24 pouces de 1680 pixels :

Ce cadre sera affiché sur $1680 * 0.8095238095238095 = 1360$ pixels et l'espace sera bien occupé.

En largeur 1050 pixels :



En largeur 1680 pixels :



Autre exemple : sur une maquette on identifie un texte de taille 32 pixels. On définira donc la taille de ce texte (exprimée en 'em') par rapport à la taille de base ; si le texte standard (ou du conteneur direct de cet élément) a une taille de 16 pixels, (1 em = 16 pixels), on définira la taille de police pour cet élément à $32/16 = 2em$ soit la règle CSS :

```
... {font-size: 2em;} /* 32/16 */
```

Sur cette même maquette composée sur 1050 pixels de large, on constate une marge autour d'un texte (*padding*) de 48 pixels.

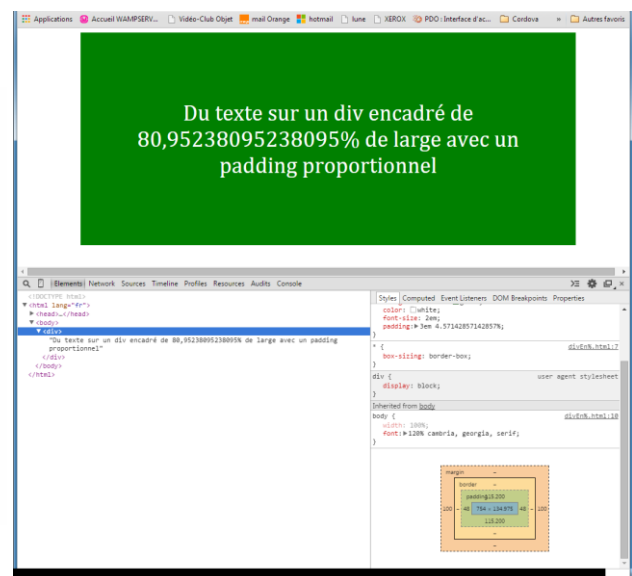
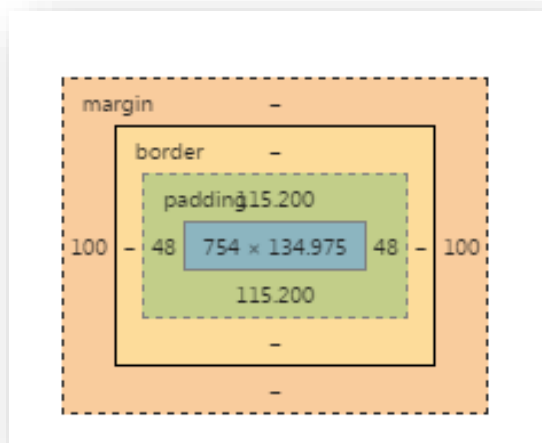
Pour un rendu RWD optimal, **on définira le *padding* en em pour le *top* et le *bottom*, et en % pour le *left* et le *right*** :

D'où la règle CSS :

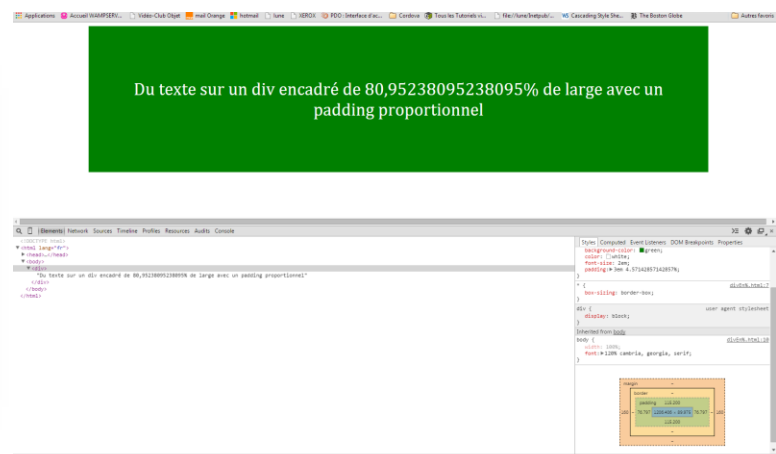
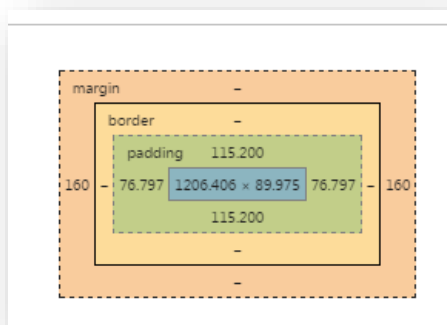
```
... {padding: 3em 4.57142857142857%;}
/* 48/16 pour top et bottom ; 48/1050 en left et right*/
```

NB : Bien entendu le navigateur effectuera le calcul et arrondira au pixel près. Mais il ne faut pas hésiter à préciser toutes les décimales (données par la calculatrice utilisée) de manière à ce que le rendu soit le plus fidèle possible selon le périphérique (1 pixel à l'écran est beaucoup plus grossier qu'un point d'imprimante, et on ne parle pas de la restitution éventuelle sur photocomposeuse d'imprimerie...). Il est de bonne pratique de toujours préciser dans un commentaire le mode de calcul pour faciliter la compréhension du code lors de la maintenance ultérieure.

En largeur 1050 pixels :



En largeur 1680 pixels :



3/ Complément sur la notion de 'em'

Le terme 'em' est issu de la typographie qui utilise toujours des polices à 'espacements proportionnels' pour lesquelles les caractères n'occupent pas tous la même place, le 'i' étant le plus étroit et le 'm' le plus large (tout comme ici, avec cette police Arial, mais ce n'est pas toujours le cas, par exemple en police Courier New, où 'i' et 'm' occupent le même espace).

Et le monde de la typographie a pris l'habitude de caractériser la taille d'une police par rapport à la taille d'un 'm', d'où cette unité de mesure 'em'.

1em = 1 fois la taille de la police de l'élément parent.

La taille d'un 'em' est relative à celle de son contexte.

Actuellement les navigateurs ont adopté une **taille par défaut de 16 pixels** et un élément <h1> a une taille par défaut de 2em, un élément <h2>, 1.5em et ainsi de suite (voir les feuilles de styles par défaut des navigateurs et la référence <http://www.iecss.com/>).

Ainsi, selon la taille par défaut de police ou le niveau de zoom effectué par l'utilisateur, les titres s'agrandissent ou se réduisent, mais toujours dans les mêmes proportions. **Reste à bien identifier le contexte (conteneur) d'un texte pour définir correctement sa taille en 'em'.**

Cela se complique parfois en cas d'imbrications d'éléments.

Exemple :

Extrait de code HTML :

```
<h1>Titre H1 contenant un <span>sous-titre</span> et la suite du H1</h1>
```

Extrait de code CSS :

```
body {
    font-size: 10px;
}

h1 {
    font-size: 3em;
    color: black;
}

h1 span {
    font-size: 2em;
    color: green;
}
```

Quelle sera donc la taille en pixel du texte h1 et celle du texte inclus dans l'élément span ?

- Pour le h1, pas d'ambiguïté, son conteneur est le body : 3*10 : 30 pixels
- Pour le span, le conteneur étant l'élément h1, il sera 2 fois plus gros que le titre (alors qu'à la lecture de la règle, on le croirait plus petit) : 2*30 = 60 pixels



Titre H1 contenant un **sous-titre** et la suite du H1

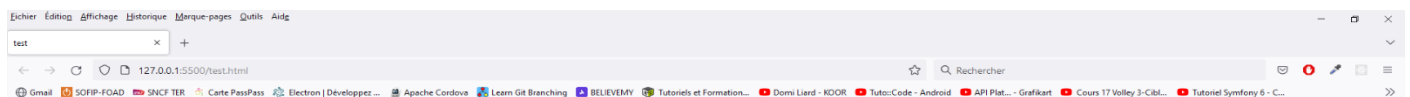
CSS offre une autre unité de mesure, cette fois-ci relative à la taille de la police de base : le **rem** (pour *root em*) :

1 rem = 1 fois la taille de la police de base (celle du body)

Ainsi en modifiant la dernière ligne de code CSS :

```
body {  
    font-size: 10px;  
}  
  
h1 {  
    font-size: 3rem;  
    color: black  
}  
  
h1 span {  
    font-size: 2rem;  
    color: green;  
}
```

on obtient bien le résultat escompté :



Titre H1 contenant un sous-titre et la suite du H1

Pour la typographie, le Responsive Design est donc basé sur ce principe : pour chaque media query, définir la taille de police par défaut puis redéfinir les tailles en 'em', par rapport à leur propre contexte (ou rem).

La taille par défaut du texte peut être définie aussi bien par l'une ou l'autre de ces règles CSS :

```
body ou html {  
    font-size :16px ;  
    font-size : 100% ;  
    font-size : 1em ;  
}
```

Afin d'assurer une bonne compatibilité entre tous les navigateurs (même les plus anciens !), il est souvent pratiqué d'initialiser toutes les valeurs par défaut des attributs CSS avant de définir son propre jeu de styles. Cela peut se réaliser très simplement en appliquant en premier un script de réinitialisation CSS.

<https://grafikart.fr/tutoriels/reset-normalize-css-1096>

normalize.css qui ne contient, quand même, pas loin de 500 lignes de code CSS !

4/ Grilles fluides

A l'aide du bon usage des dimensions CSS proportionnelles, la mise en page va donc pouvoir s'adapter finement aux caractéristiques du périphérique de restitution.

A l'aide des media queries CSS, on peut définir des mises en page variées selon des plages de dimensions.

Ces deux techniques CSS, **mesures relatives** et **media queries**, se complètent donc pour **assurer le principe de 'grille fluide'** qui tend à conserver la même mise en page dans une même plage et atténuer les changements de mise en page d'une plage à l'autre.

Par exemple :

```
@media screen {  
  nav ul {  
    width: 100%;  
  }  
}  
  
@media screen and (min-width :1051px) {  
  nav ul li a {  
    font-size: 1.4em;  
  }  
}  
  
@media screen and (min-width :641px) and (max-width :1050px) {  
  nav ul li a {  
    font-size: 1.25em;  
  }  
}  
  
@media screen and (max-width :640px) {  
  nav ul li a {  
    font-size: 1.1em;  
  }  
}
```

Aidera à assurer que les liens de navigation occupent bien une ligne du périphérique, aussi bien sur écran de PC, sur tablette ou sur smartphone.

IX Images fluides

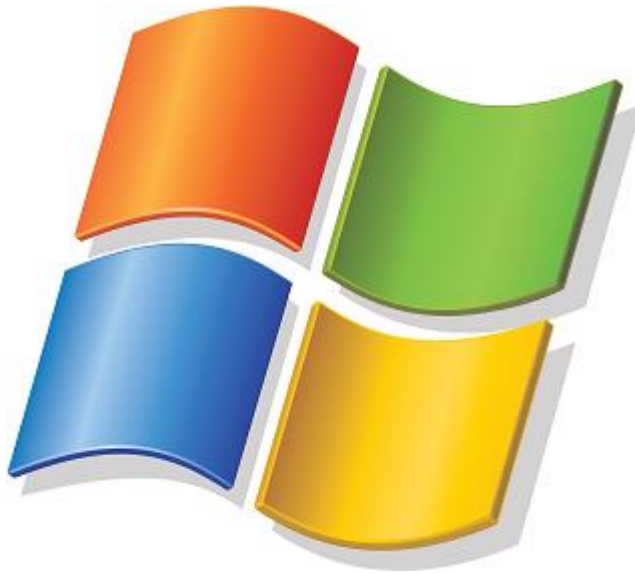
1/ Zoom sur les images

Pour assurer une bonne adaptation de la page aux caractéristiques physiques du navigateur, il est nécessaire que les images effectuent automatiquement un effet de zoom selon la taille (proportionnelle) de leur conteneur.

Pour cela la règle CSS suivante suffit.

```
img {  
    max-width: 100%;  
    height: auto;  
}
```

Mais si une image 'bitmap' (comme les .gif, .jpg ou .png) supporte très bien le rétrécissement, elle devient bien vite inesthétique (par 'effet d'escalier' et 'floutage') lors de son agrandissement.



A taille normale :



A 25% :

A 500% :



Il serait donc préférable de définir les images en taille maximale afin d'assurer un bon rendu sur grand écran et que les périphériques plus petits effectuent des réductions.

Mais le poids d'une image (en octets, KO, MO) est proportionnel à ses dimensions en pixels et le temps de transfert entre serveur Web et périphérique est lui aussi proportionnel au volume de l'image ; comble de malheur, les terminaux mobiles sont actuellement desservis par des réseaux beaucoup plus lents que ceux utilisés pour les PC de bureau !

Il faut donc trouver un compromis entre poids de l'image et niveau de détail, surtout si l'on veut s'adapter aux terminaux mobiles.

Pour les grands écrans, on peut déjà ajouter la règle CSS `max-width : yypx;`, et, pour les petits, la règle symétrique `min-width : zzpx;` de manière à limiter les altérations de l'image.

Au-delà, si l'on refuse les compromis (cas d'un site hautement graphique comme la publication de photographies), on peut toujours prévoir chaque image dans plusieurs dimensions et servir dynamiquement l'image adaptée au mieux au périphérique demandeur ; mais cela nécessite du traitement dynamique côté serveur Web (en PHP, JSP...) qui va bien au-delà de nos préoccupations purement HTML/CSS.

Il existe aussi des outils/Framework côté serveur comme 'Adaptive Images' qui réalisent automatiquement un travail de conversion des images et mise en cache sur le serveur après détection des caractéristiques de l'écran du visiteur.

<http://adaptive-images.com/>

2/ Habillage de l'image par le texte

HTML offre depuis toujours la possibilité d'habiller une image par le flux HTML suivant grâce à l'attribut `align` de la balise ``, et il reste toujours possible d'interrompre un habillage en insérant une balise `<br clear="xx" />`.

Alors pourquoi s'en priver ? Parce que cette méthode fige la présentation dans le code HTML et un des buts de CSS est de pouvoir changer le 'look' d'un site à tout moment en modifiant simplement la feuille de styles.

Oublier les attributs HTML align des images au profit de la propriété CSS float.

Comme CSS permet de reproduire tout ce que fait HTML et même plus, cette technique d'habillage est disponible par CSS, et pas seulement pour les images, car les propriétés `float` et `clear` s'appliquent à tous les éléments. C'est ainsi que l'on place par CSS un bloc de navigation à côté du contenu principal ou que l'on insère un encadré dans un bloc pour un affichage sur écran de PC, et que l'on place tous les éléments en séquence, dessus-dessous, sur écran mobile.

X Mise en page utilisant un système de grille CSS

1/ Principe de base

Il s'agit encore ici d'un principe issu du monde de l'édition. Chaque publication (quotidien, magazine) adopte une mise en page respectée, avec des variantes, sur toutes les pages. Cette présentation est basée sur un colonage régulier des pages (5 colonnes, 8 colonnes...). Pour rompre la monotonie, un texte, une image ou un encadré peut parfois courir sur plusieurs colonnes sans remettre en cause le colonage de base.

En Responsive Design, les graphistes ont rapidement adapté ce principe afin d'assurer une certaine homogénéité dans les diverses présentations selon les largeurs d'écrans ; les framework CSS comme Bootstrap ou Foundation reposent sur un système de grille.

Voici le principe du système de grille colonée :

Quel que soit le périphérique, la page est composée sur une grille fictive de 12 colonnes. Pour chaque media query, des classes de styles sont prévues pour couvrir de 1 à 12 colonnes. Les éléments HTML de la page sont affectés de 1 ou plusieurs classes de styles et ils occuperont un nombre plus ou moins grand de colonnes en fonction de la classe de style appliquée par le navigateur selon le media query correspondant au périphérique.

Ce système de grille par l'utilisation des classes égratigne la règle de ne pas surcharger les attributs css class ou Id défendue jusqu'à maintenant.

L'exercice suivant décompose ce processus pour mieux comprendre le principe.

2/ Exercice : Vers une grille adaptative

Il s'agit tout d'abord de simuler un menu qui s'affiche sur toute la largeur de l'écran ; chaque ligne du menu propose de 1 à 12 items selon la place disponible. On identifie 3 largeurs types :

- Moins de 640 pixels : chaque item occupe toute la largeur (sauf pour la dernière ligne)

Grille Responsive Design - Exemple sur un menu	
Grille Responsive	
Menu A	
Menu B	
Menu C	
Menu D	
Menu E	
Menu F	
Menu G	
Menu H	
Menu I	
Menu J	
Menu K	
Menu L	
Menu M	
Menu N	
Menu O	
Menu P	
Menu Q	Menu R

- entre 640 et 1024 pixels : on affiche entre 2 et 6 items par ligne

Grille Responsive Design - Exemple sur un menu

Grille Responsive

Menu A			Menu B		
Menu C			Menu D		
Menu E			Menu F		
Menu G	Menu H	Menu I	Menu J	Menu K	Menu L
Menu M	Menu N	Menu O	Menu P	Menu Q	Menu R

- et plus de 1024 pixels : on affiche entre 2 et 12 items par ligne sur 90% de la largeur

Grille Responsive Design - Exemple sur un menu

Grille Responsive

Menu A						Menu B					
Menu C			Menu D			Menu E			Menu F		
Menu G	Menu H	Menu I	Menu J	Menu K	Menu L	Menu M	Menu N	Menu O	Menu P	Menu Q	Menu R

La Structure HTML de cette portion de page est simplement basée sur une classique imbrication d'éléments `<div>` :

```
<body>
  <div class="menu">
    <h2>Grille Responsive</h2>
    <div>Menu A</div>
    <div>Menu B</div>
    <div>Menu C</div>
    <div>Menu D</div>
    <div>Menu E</div>
    .....
    <div>Menu Q</div>
    <div>Menu R</div>
  </div>
</body>
```

Un élément *div* étant par défaut un '*block*', chaque option de menu s'étend de la marge de gauche à la marge de droite (**width :100%** ; implicite) :

Feuille de styles pour mobiles

On adopte une démarche Mobile First : il s'agit donc de définir tout d'abord la feuille de styles pour écrans étroits contenant aussi les styles communs aux diverses présentations.

Fichier CSS general.css :

```
*, *:before, *:after {
  box-sizing: border-box;
}
body {
  margin: 0px;
  padding: 0px;
  font-family: sans-serif;
}
div.menu>div {
  text-align: center;
}
/* présentation de tous les items de menus */
```

```

.col-xs-12, .col-xs-8, .col-xs-6, .col-xs-4, .col-xs-3, .col-xs-2, .col-xs-1,
.col-md-12, .col-md-8, .col-md-6, .col-md-4, .col-md-3, .col-md-2, .col-md-1,
.col-l-12, .col-l-8, .col-l-6, .col-l-4, .col-l-3, .col-l-2, .col-l-1 {
    padding: 0 10px;
    border: 1px solid blue;
    margin: 1px 0px;
}

```

On prévoit ensuite systématiquement l'ensemble des classes de styles pour affichage des items de menus sur 1 à 12 colonnes (ici, on se limite à 1, 2, 3, 4, 6, 8 et 12 colonnes) :

Fichier CSS mobile.css

```

/* colonnage Mobile (démarche Mobile First) :
-----
grille de 12 éléments ==> styles pour affichage sur 12, 8, 6, 4, 3, 2, 1 colonnes */
/* attributs communs à toutes les présentations de menu */
.col-xs-12, .col-xs-8, .col-xs-6, .col-xs-4, .col-xs-3, .col-xs-2, .col-xs-1 {
    background-color: lightblue;
}
/* affichage sur 12 colonnes : 12/12 */
.col-xs-12 {
    width: 100%;
}
/* affichage sur 8 colonnes : 8/12 */
.col-xs-8 {
    width: 66.6666%;
}
/* affichage sur 6 colonnes : 6/12 */
.col-xs-6 {
    width: 50%;
}
/* affichage sur 4 colonnes : 4/12*/
.col-xs-4 {
    width: 33.3333%;
}
/* affichage sur 3 colonnes : 3/12*/
.col-xs-3 {
    width: 25%;
}
/* affichage sur 2 colonnes : 2/12*/
.col-xs-2 {
    width: 16.6667%;
}
/* affichage sur 1 colonne : 1/12*/
.col-xs-1 {
    width: 8.3333%;
}

```

On affecte à chaque option de menu la classe de style qui permet de s'étendre sur toute la largeur :

```
<div class="col-xs-12 ">Menu A</div>
```

Mais pour les 2 dernières options, on souhaite les placer côte-à-côte :

```
<div class="col-xs-6">Menu Q</div>
<div class="col-xs-6">Menu R</div>
```

On relie la feuille de styles à la page HTML, grâce à une balise <link> sans attribut media, sans aucune restriction de largeur. On doit alors obtenir le résultat voulu pour petits écrans, quel que soit la largeur d'affichage.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="general.css">
  <link rel="stylesheet" href="mobile.css">
```

Feuille de styles pour écrans 'medium'

Pour les écrans de taille moyenne, typiquement les écrans standards de PC, on établit une nouvelle feuille de styles assez similaire. L'approche Mobile First permet de préciser uniquement ce qui est nouveau ou de modifier un style existant si nécessaire, pour les largeurs d'écrans supérieures à 639 pixels.

Fichier CSS medium.css

```
.col-md-12, .col-md-8, .col-md-6, .col-md-4, .col-md-3, .col-md-2, .col-md-1 {
  background-color: orange;
}
/* affichage sur 12 colonnes */
.col-md-12 {
  width: 100%;
}
/* affichage sur 8 colonnes : 8/12 */
.col-md-8 {
  width: 66.6666%;
}
/* affichage sur 6 colonnes : 6/12 */
.col-md-6 {
  width: 50%;
}
/* affichage sur 4 colonnes : 4/12 */
.col-md-4 {
  width: 33.3333%;
}
/* affichage sur 3 colonnes : 3/12 */
.col-md-3 {
  width: 25%;
```



```

}
/* affichage sur 2 colonnes : 2/12 */
.col-md-2 {
  width: 16.6667%;
}
/* affichage sur 1 colonne : 1/12 */
.col-md-1 {
  width: 8.3333%;
}

```

Bien sûr, les feuilles de styles ne sont pas vraiment optimisées puisque de nombreuses règles de styles sont similaires au cas précédent, mais rappelons que le but est de définir un cadre général. Reste à appeler la classe de styles dans le code HTML ; toute l'astuce réside dans le fait **d'AJOUTER ou non une classe** pour chaque item de menu ; selon la feuille de styles sélectionnée par le navigateur, L'UNE OU L'AUTRE des classes sera appliquée :

```
<div class="col-xs-12 col-md-6">Menu A</div>
```

Ici, on choisit la classe **col-md-6** pour les 6 premiers items qui se coulent alors 2 par ligne, puis on appelle la classe **col-md-2** pour les items suivants qui se coulent 6 par ligne.

Ne pas oublier de relier la feuille de styles à la page HTML grâce à la balise <link> avec l'attribut media qui va bien :

```
<link rel="stylesheet" href="milieu.css" media="only screen and (min-width:640px)">
```

Feuille de styles pour écrans larges

De même, établir une nouvelle feuille de styles pour les écrans larges, affichant au moins 1024 pixels en largeur ; définir les différentes largeurs de colonnes, préciser que le menu n'occupe que 90% de la largeur disponible et définir la couleur de fond :

```

div.xl {
  /* menu sur 90% seulement */
  width: 90%;
  margin: 0 auto;
}
.col-l-12, .col-l-8, .col-l-6, .col-l-4, .col-l-3, .col-l-2, .col-l-1 {
  background-color: lime;
}
/* affichage sur 12 colonnes */
.col-l-12 {
  width: 100%;
}
/* affichage sur 8 colonnes : 8/12 */
.col-l-8 {
  width: 66.6666%;
}
/* affichage sur 6 colonnes : 6/12 */
.col-l-6 {
  width: 50%;
}
/* affichage sur 4 colonnes : 4/12 */
.col-l-4 {
  width: 33.3333%;
}
/* affichage sur 3 colonnes : 3/12 */

```

Il faudra ajouter dans le code HTML l'appel de cette classe de styles ; comme elle est absente des autres feuilles de styles, elle sera ignorée sur écrans medium et mobile.

```

.col-l-3 {
  width: 25%;
}
/* affichage sur 2 colonnes : 2/12 */
.col-l-2 {
  width: 16.6667%;
}
/* affichage sur 1 colonne : 1/12 */
.col-l-1 {
  width: 8.3333%;
}

```

Il reste à ajouter l'appel de la classe voulue pour chaque item de menu (2, 4 ou 12 items par ligne) et à ajouter l'appel de la classe `xl` dans l'élément contenant le menu. Pour finir, relier la feuille de styles à la page HTML

Petit extrait du HTML final :

```

<body>
  <div class="menu xl">
    <h2>Grille Responsive</h2>
    <div class="col-xs-12 col-md-6 col-l-6">Menu A</div>
    <div class="col-xs-12 col-md-6 col-l-6">Menu B</div>
    <div class="col-xs-12 col-md-6 col-l-3">Menu C</div>
    <div class="col-xs-12 col-md-6 col-l-3">Menu D</div>
    <div class="col-xs-12 col-md-6 col-l-3">Menu E</div>
    <div class="col-xs-12 col-md-6 col-l-3">Menu F</div>
    <div class="col-xs-12 col-md-2 col-l-1">Menu G</div>
    <div class="col-xs-12 col-md-2 col-l-1">Menu H</div>
    <div class="col-xs-12 col-md-2 col-l-1">Menu I</div>
    <div class="col-xs-12 col-md-2 col-l-1">Menu J</div>
  </div>

```

Tout cela fonctionne comme souhaité. Pour bien comprendre le mécanisme et l'intérêt de ce système, ajoutons une dernière option au menu, appelant une seule classe :

```
<div class="col-xs-12">Menu S</div>
```

Testez.

Sur petit écran, l'option se coule bien sur toute la largeur. Et elle garde le même comportement pour les écrans medium et larges ! Observez comment le navigateur applique les styles grâce au débogueur intégré.

Simulation de mise en page

Un menu adaptatif, c'est bien ; mais une page à géométrie variable, c'est mieux !
Ajoutons une structure de page HTML5 très simplifiée sous le menu :

```
<h2>Simulation de mise en page</h2>
<div class="">
  <header class="col-xs-12">
    Entete
  </header>
  <nav class="col-md-4">
    Menu
  </nav>
  <section class="col-md-8">
    <article class="col-l-8">
      Article
    </article>
    <aside class="col-l-4">
      Aside
    </aside>
  </section>
</div>
```

Traduction :

header : s'étend toujours sur 12 colonnes

nav : s'étend sur 4 colonnes à partir des écrans medium ; en petit écran, se comporte comme `col-xs-12` car c'est un *block* et sa largeur n'est pas définie

section : s'étend sur 8 colonnes à partir des écrans medium, sinon, sur 12 colonnes

article : s'étend sur 8 colonnes à partir des écrans larges, sinon, sur 12 colonnes

aside : s'étend sur 4 colonnes sur écrans larges, sinon, sur 12 colonnes

Testez en observant bien les classes appliquées.

Cette fois, vous aurez quasiment reproduit le système de grilles intégré à Bootstrap !

Bilan :

- On dispose maintenant d'une large bibliothèque de classes permettant de composer les portions de page qui s'étendent sur 1 à 12 colonnes et dont l'apparence reste totalement personnalisable selon la largeur d'écran ;
- Pour chaque élément concerné par un design 'Responsive', on appelle dans le code HTML une ou plusieurs classes, chacune étant adaptée à un media query particulier ;
- On appelle tout d'abord la classe de styles correspondant à la plus petite largeur puis, si la présentation doit varier pour des largeurs supérieures, on ajoute les appels des classes nécessaires ;
- On définit tout d'abord le jeu de styles nécessaire pour affichage sur mobile puis on complète/modifie avec les jeux de styles pour les autres périphériques ; ainsi, un mobile ne chargera que le strict nécessaire ;
- Avec ce principe, la structure HTML reste indépendante de la présentation par CSS et on obtient bien une mise en page adaptative mais il est nécessaire d'assembler dans le code HTML les (nombreuses) classes de styles.

XI Framework CSS et Préprocesseurs

1/ Mise en page avec un Framework CSS

Plutôt que de construire soi-même tout cet ensemble de classes de base, il peut être très tentant d'utiliser un Framework CSS existant.

Intérêts :

- Les navigateurs ont des comportements parfois très différents malgré leur lente convergence vers les standards. Les Framework sont *cross-browser*, c'est à dire que la présentation est similaire quel que soit le navigateur utilisé et d'une parfaite compatibilité.
- Les Framework CSS font gagner du temps de développement parce qu'ils offrent les fondations de la présentation ; reste à personnaliser le graphisme (couleurs, polices, encadrés...).
- Les Framework CSS normalisent la présentation en proposant un ensemble homogène de styles *a priori* esthétiques et fonctionnels, ce qui est très pratique quand le développeur doit assurer lui-même la présentation graphique.
- Les Framework CSS proposent en général un système de grille pour faciliter le positionnement des éléments.
- Les Framework CSS offrent souvent des éléments complémentaires : boutons esthétiques, barres de navigation, menus déroulants...
- Les Framework CSS prennent généralement en compte les nouveaux moyens de visualisation du Web (smartphones, tablettes...) et sont mis à jour très rapidement.

Inconvénients :

- Un Framework CSS représente un volume de code conséquent qu'il reste à interpréter (plus de 100Ko et pas moins de 6500 lignes de code CSS pour la feuille de styles Bootstrap !)
- Pour utiliser efficacement un Framework il faut bien le connaître, ce qui implique un temps d'apprentissage non négligeable (et en continu).
- La normalisation de la présentation peut devenir lassante pour un infographiste en lissant les effets visuels au détriment de la création originale.
-

Aujourd'hui les 2 Framework CSS les plus utilisés sont **Twitter Bootstrap** et **Foundation**. Il en existe plein d'autres.

2/ Préprocesseurs CSS





1) Un préprocesseur CSS, pourquoi faire ?

Quand un (bon) développeur observe son travail d'écriture d'une feuille de styles CSS, il est contrarié par le fait que du code se répète, parfois avec des variantes mineures, aussi bien dans les noms de classes (voir l'exercice sur les grilles) que dans des valeurs de propriétés.


*La raison principale de sa contrariété vient du fait que le langage CSS ne supporte pas les notions de **variable**, de **test**, ni de **boucle** qui constituent les 3 'mamelles' de tout langage de programmation.*

Les préprocesseurs CSS ont été conçus pour réparer ce 'mal' ! Ce sont donc des outils à destination de développeurs confrontés à l'écriture de feuilles de styles, plutôt que des aides aux intégrateurs Web, plus infographistes que développeur. Pour s'en convaincre, il suffit de jeter un œil sur les commentaires en ligne concernant les préprocesseurs CSS...

Aujourd'hui, il existe plusieurs préprocesseurs CSS dont :

- Less  disponible sur : <http://lesscss.org/>
- Sass   disponible sur : <http://sass-lang.com/>
- Stylus  disponible sur : <http://learnboost.github.io/stylus/>

Chacun a son fonctionnement et sa syntaxe spécifique mais ils offrent tous les mêmes fonctionnalités de base. Sass est maintenant complété par le framework Compass

 (disponible sur <http://compass-style.org/>), et il est de loin le plus populaire parmi les développeurs.

2) Découverte de Sass/Compass

Avant d'explorer Sass/Compass, voici déjà ce qu'on peut en attendre :

- L'écriture simplifiée en langage Sass de 'modèles' de feuilles de styles et la génération du code CSS final à adresser au client Web ;
- Un langage de programmation (Sass) très proche du CSS ;
- L'introduction de variables dans le code (identifiées par le préfixe \$) ;
- La possibilité d'écrire des boucles de type '*pour i allant de x à y*' permettant de générer des portions de code CSS similaires ;
- La possibilité de définir ses propres fonctions réutilisables, éventuellement paramétrées (notion de '*mixins*') ;
- La mise à disposition de fonctions intégrées au langage Sass, tout spécialement destinées à l'écriture de code CSS (comme `darken()` ou `lighten()` pour faire varier les valeurs d'une couleur) ;
- La mise à disposition d'une vaste bibliothèque de mixins inclus dans Compass qui facilite le travail comme la génération des variantes de syntaxe préfixées pour les propriétés CSS mal stabilisées (la simple instruction `@include box-shadow` ; suffit à générer toutes les variantes connues pour l'ombrage de boîtes) ;
- Le support, grâce à Compass, de l'héritage entre classes de styles (mixin `@extend`) ;
- La génération automatique et en temps réel du code CSS commenté, grâce à Compass, avec insertion éventuelle d'un message d'erreur affiché automatiquement dans la page Web ;
- La génération de versions compressées (*minification*) des feuilles de styles pour en alléger le poids et le temps de chargement ;

Explorez maintenant comment simplifier l'écriture d'un code CSS typique grâce à Sass/Compass en étudiant la vidéo de GrafikArt « Framework CSS Compass » :

<http://www.grafikart.fr/tutoriels/html-css/framework-css-compass-140>

Pour aller plus loin dans la mise en œuvre de Sass/Compass dans une logique Responsive Design, étudiez la vidéo « Créer une grille Responsive » du même GrafikArt :

<http://www.grafikart.fr/tutoriels/html-css/grille-responsive-css-498>

Et oui, c'est bien sur ce genre de code CSS qu'un préprocesseur CSS prend toute sa valeur !