Formation Responsive web design

Objectif:

Apprendre à maitriser le concept et la méthode Responsive Web Design afin de créer des sites multi-plateformes, fonctionnant sur ordis, iphone, ipad, android, smartphone, tablettes, tv etc.

Prérequis:

Connaissances de base en XHTML / CSS conseillées

Les terminaux

Maudit soit celui qui a inventé les smartphones, car depuis, l'univers du web a été totalement chamboulé.Comme si le web designer ne pouvait trouver de repos dans son métier :

- Des années à chercher des astuces pour s'afficher correctement sur 2 (3?) navigateurs

- Puis d'un coup, l'iPhone infléchit tout le marché.

- Aujourd'hui, la fragmentation des appareils d'affichage du web est telle que l'on est tenté de créer autant de sites que de périphériques.

- Des nouveaux périphériques ne cessent de voir le jour : google glass et autres consoles de jeu ou téléviseur super haute déf.

Il nous reste alors deux solutions, soit nous arrêtons dès maintenant ce métier, soit nous devenons flexibles, et nous apprenons à créer des sites web responsive design qui répondent à nos besoins (et à ceux des mobinautes).

Commençons déjà par faire un tour du côté des résolutions !

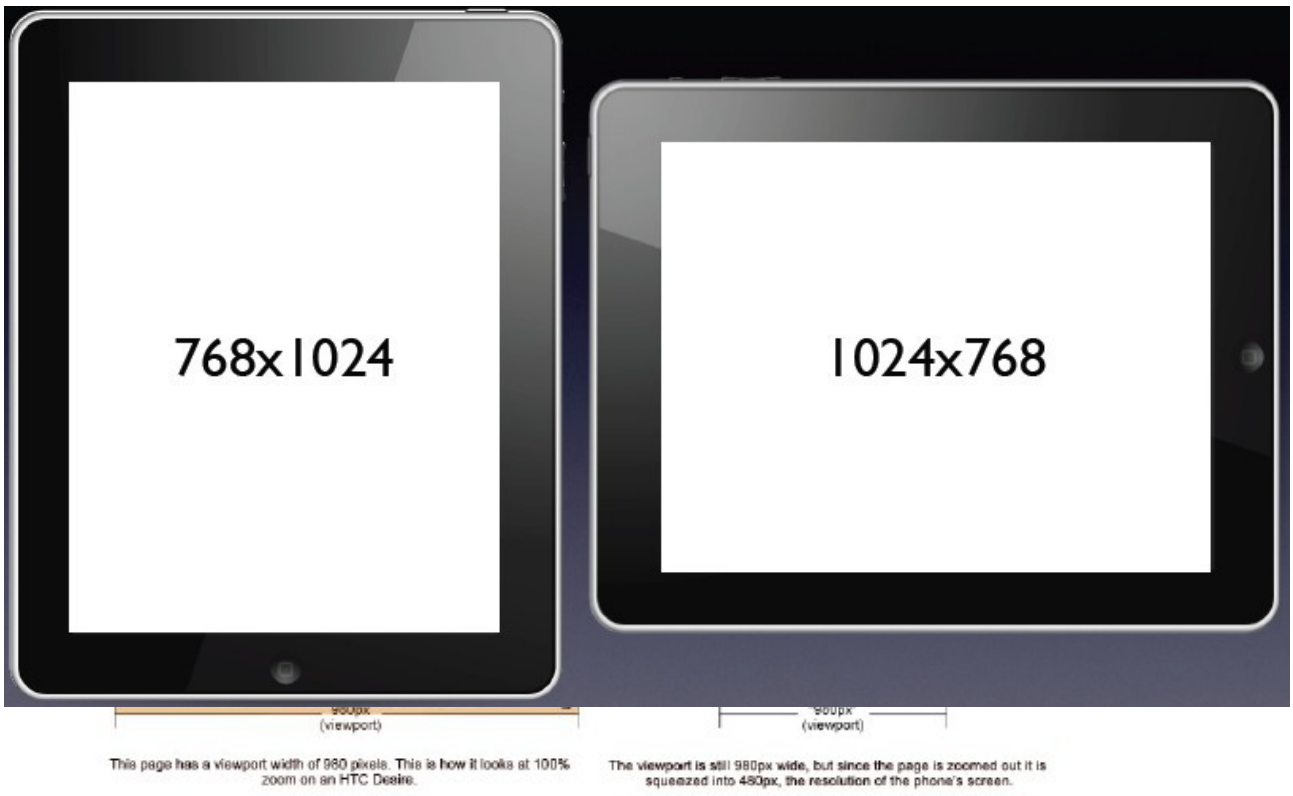
Les résolutions Web, mobile, tablette et TV

Avec l'avènement du web mobile, le nombre de résolutions à gérer a grimpé en flèche. Sans compter que les terminaux mobiles aujourd'hui permettent d'afficher un nombre de pixels bien supérieurs à celui de la taille des écrans.

Et Qu'ils disposent pour la plupart d'un mode portrait et d'un mode paysage...

Résolutions Web, mobile, tablettes, TV

- Dimensions en "surface réelle"
  - 320x480px pour l'iPhone 3
  - 640x960px pour l'iPhone 4
  - 640x1136px pour l'iPhone 5
  - 768x1024px pour l'iPad 2
  - 1536x2048px pour l'iPad 3
- Dimensions en pixels CSS
  HTML : device-width / device-height
  JS: screen.width / screen.height
  - 320x480px pour l'iPhone 3
  - 320x480px pour l'iPhone 4
  - 320x568px pour l'iPhone 5
  - 768x1024px pour l'iPad 2
  - 768x1024px pour l'iPad 3



Maintenant que nous avons à quel point nous sommes dans le pétrin, je vous propose de voir de quelle façon nous allons nous en sortir.

Démarche de conception

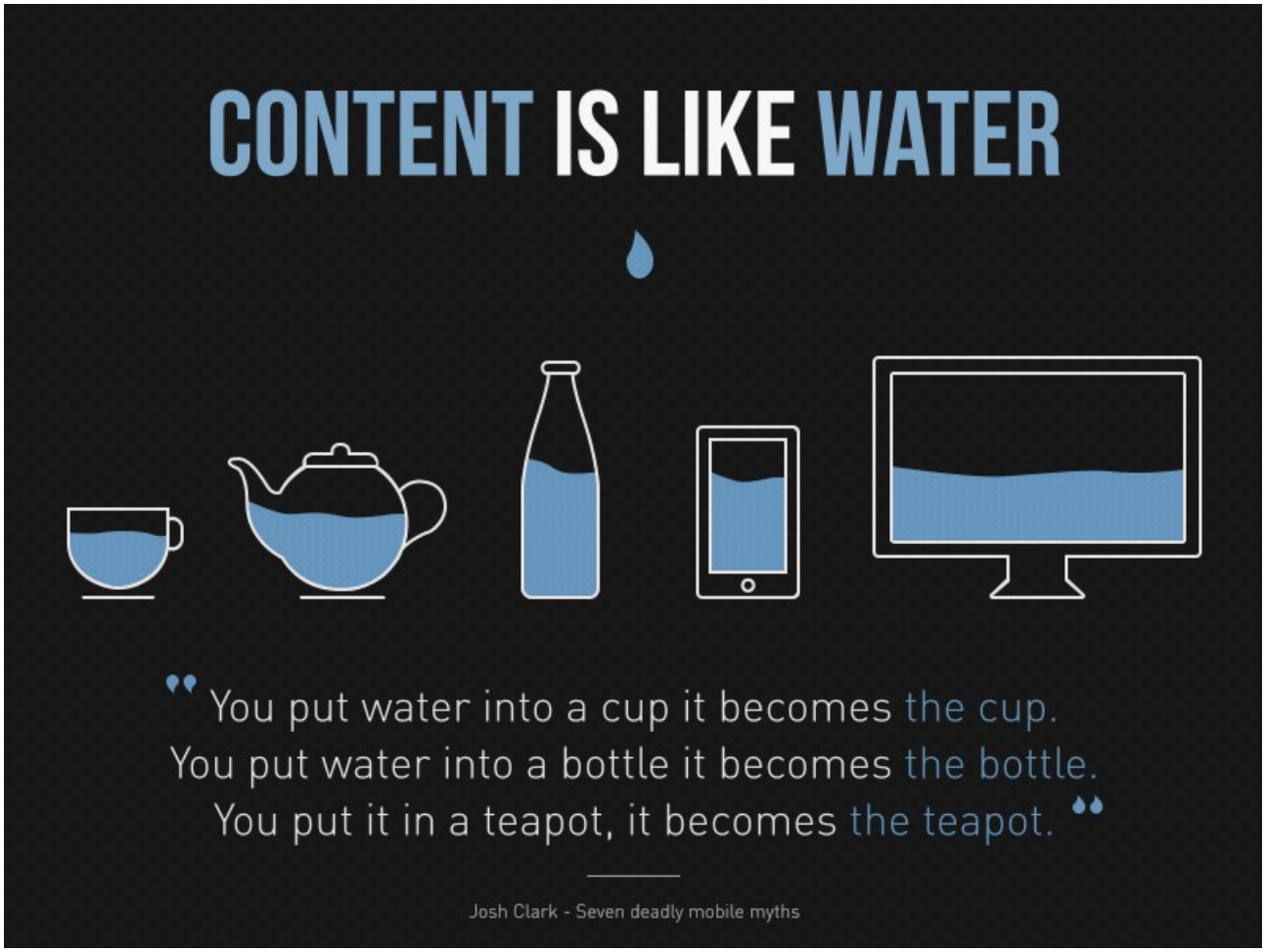
La philosophie Responsive

Mais que veulent dire ces termes au juste ? Responsive Web Design ? Même pour les plus anglophones ces termes peuvent paraître obscurs, voyons ce que dit Wikipédia à ce sujet (qui en parle dans les termes bien franco-français de "site web adaptatif"):

"Un site web adaptatif [...] est une notion de conception de sites web qui regroupe différents principes et technologies dans laquelle un site est conçu pour offrir au visiteur une expérience de consultation optimale facilitant la lecture et la navigation.

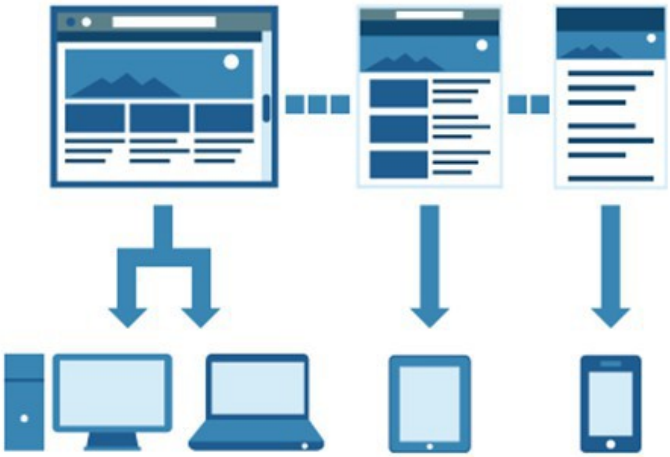
L'utilisateur peut ainsi consulter le même site web à travers une large gamme d'appareils (moniteurs d'ordinateur, smartphones, tablettes, TV, etc.) avec le même confort visuel et sans avoir recours au défilement horizontal ou au zoom avant/arrière sur les appareils tactiles notamment, manipulations qui dégradent considérablement l'expérience utilisateur."

C'est un peu obscur, je vous propose donc de visualier cela sous la forme d'un schéma:



Idéalement aujourd'hui, il faudrait que le contenu s'adapte en permanence au device sur lequel il est visionné, un peu come l'eau, voilà ce qu'est- la science du Responsive Web Design, l'adaptation du contenu au contenant, l'ergonomie au premier plan.

Dans le cadre d'un site web ça pourrait donner cela :



On voit bien ici que le contenu s'est réorganisé en fonction du device sur lequel il est visionné. Et qu'il faut également tenir compte des interactions utilisateur disponibles en fonction du device (touch, click, clavier etc ...)

Voyons maintenant de quelle façon nous allons aborder la conception d'un site responsive.

Une évolution des techniques

Hier

1. Dimensions connues : 640x480 / 800x600 / 1024x768
2. Largeur fixe
3. Les développeurs prévoyaient un site pour écran large, testaient sur les différents navigateurs, et pensaient ensuite à tester sur certains écrans plus petits

Aujourd'hui

1. On pense aux différences d'affichage dès la maquette
2. On fait en sorte de proposer un expérience acceptable pour tous
3. On améliore l'expérience pour ceux qui ont la possibilité d'en profiter

A l'ère du responsive web design, on aura tendance à penser son site web ou son application pour le mobile en premier. On pensera à optimiser le poids des images, des fonts, des vidéos etc.. en fonction du device ciblé.

On va donc pouvoir mettre en évidence quelques techniques dont les suivantes :

- Une mise en page par grilles CSS fluides, en %. Utile pour faire en sorte que la page, la disposition des éléments, et la taille de texte s'adapte aux dimensions variables des machines
- Le recours aux media-queries CSS3. Utile pour opérer une transformation de la disposition des éléments ajuster dimensions et marges masquer et afficher des éléments.
- Images et éléments multimédias fluides. Utile pour ajuster et optimiser les tailles et poids des éléments multimédia
- L'amélioration progressive consiste à créer des pages web multicouches.

A minima, tout le monde peut voir et utiliser le contenu, et le temps de chargement est optimisé

- Les utilisateurs équipés de navigateurs plus récents obtiennent des couches supplémentaires de styles et d'interactivité qui améliorent l'expérience

- Privilégier l'ordre : Mobile basique -> Smartphone -> Tablette -> Ordinateur

- Graceful degradation: Lorsque certaines fonctionnalités ne sont pas compatibles sur tous les navigateurs, l'idée de "Graceful degradation" consiste à recommander l'utilisation de ces fonctionnalités quand c'est possible et de prévoir une expérience acceptable quand ça ne l'est pas.

Par exemple: certaines propriétés CSS3 comme les transformations 3D, ne sont pas gérées par IE10, en le prenant en compte, on peut prévoir une alternative en javascript, ou une navigation différente afin que l'expérience soit acceptable.

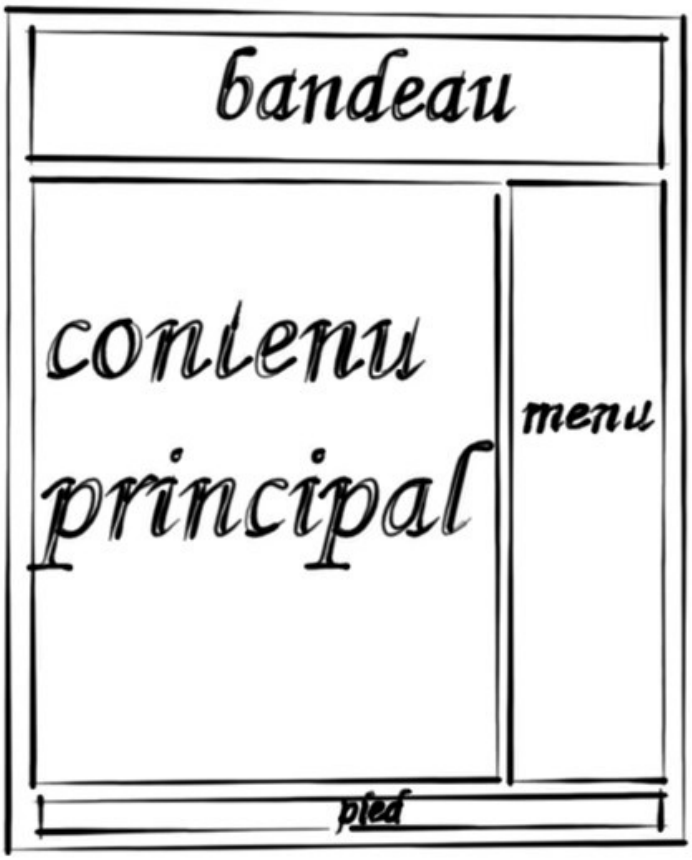
La mise en place de ces différentes techniques nous amène inévitablement à penser notre application d'une façon différente, centrée en priorité sur la version mobile. Il s'agit d'une façon de faire reconnue dans le monde du web responsive, tellement qu'on lui a même donné un nom: **Mobile First**.

Cette "technique" ou "philosophie" appellez ça comme vous voulez, est centrée sur l'application de l'ensemble de ces principes.

Preparer un site flexible et créer un plan de test

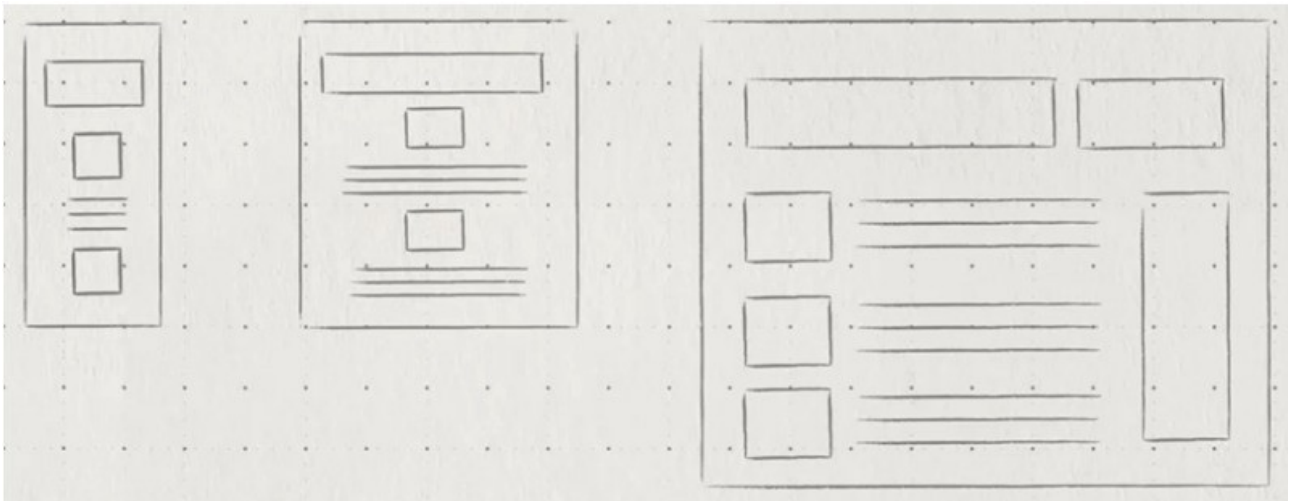
- Définir des schémas pour différentes résolutions en se posant les questions suivantes :

1. Quels éléments sont importants ?
2. Quels léments sont amenés à se déplacer (blocs de textes, etc.) ?
3. Quels éléments sont amenés à changer (menu, etc.) ?
4. Quels éléments et fonctionnalités peuvent/doivent être supprimés pour certaines7 versions du site (publicités, chat, géolocalisation, etc.) ?



Avant de créer un site, il faut l'imaginer et le schématiser. On peut commencer par la page version large, puis on décline en tablette et mobile. Mais il est conseillé et fortement recommandé de commencer par la version mobile .

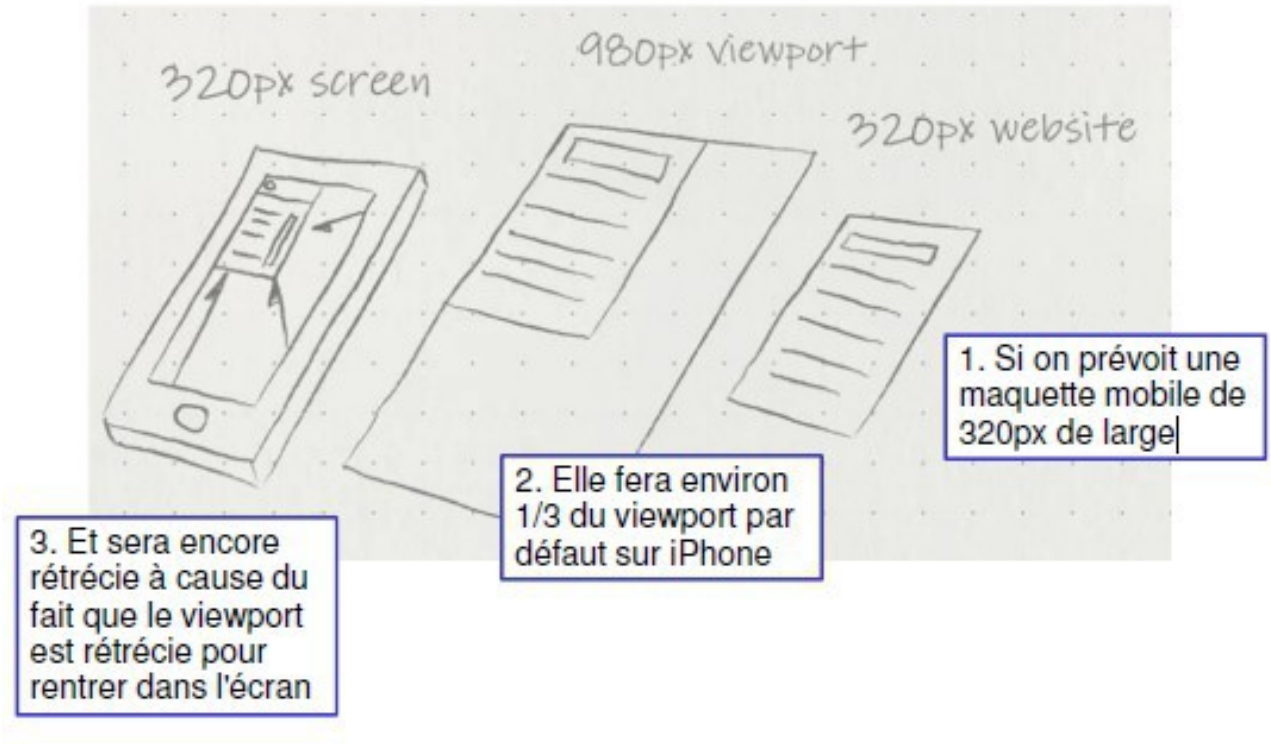
Décliner son site :



Analyser la structure de la page, et identifier les éléments à changer
on peut passer de 3 colonnes à 1000px de large, à 1 colonne de 320px de large par exemple, ou bien cacher des pubs, modifier l'aspect d'un menu, etc.

Préparer son site pour les terminaux

Le Viewport:



Le viewport est la zone d'affichage, sur les ordis, c'est la fenêtre du navigateur, mais sur les mobiles, elle est plus grande que l'écran !

Par défaut, sur un mobile, son échelle change afin d'afficher l'ensemble de la page, on "dézoome ".

On peut spécifier programmatiquement la largeur du viewport

en html

```
<meta name="viewport" content="width=320">
```

en css

```
@viewport
{
	width:320px;
}
```

On peut la fixer à la largeur de l'appareil

en html

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

en css

```
@viewport
{
		width:device-width;
}
```

On peut fixer le zoom initial, en le mettant à 1, le contenu s'affiche à 100%

en html

```
<meta name="viewport" content="initial-scale=1">
```

en css

```
@viewport
{
		zoom:1;
}
```

Afin de palier au différents affichages suivant les périphériques, une bonne pratique consiste à définir la meta viewport dans le <head>. Certains recommandent de forcer aussi le width à device-width, d'autres de forcer seulement l'échelle, d'autres de forcer les deux. Cette méthode évite des bugs sous iOS.

Exemple:

en html

```
<meta name="viewport" content="initial-scale=1"> ou
<meta name="viewport" content="width=device-width, initial-scale=1">
```

en css

```
@viewport
{
        zoom:1;
}
```

Les Media Queries

Elles permettent de modifier les css d'un page en fonction du type de média (résolution) et de différents critères (fonctionnalités), comme ses dimensions.

Process : identifier les points de rupture du design, puis générer les CSS adaptées et utiliser les mediaqueries en conséquence.

1. Préparer les différents blocs de codes en fonction des points de ruptures.
2. Préciser les modifications (largeurs, emplacements, etc.).
3. Simplifier le CSS en regroupant les éléments

Points de bascule types: Les points de bascule représentent le moment où la disposition d'un site doit changer. Cela dépend généralement de la largeur du viewport, on peut utiliser par exemple les points de bascule suivants :

- Entre 0 et 480 px de large : mobiles (maquette : 320px)
- Entre 481 et 768 px de large : tablette (maquette : 768px)
- Entre 769 et 1279 px de large : pc (maquette : 1000px)
- Dès 1280 px de large : ordinateur à écran large (maquette : 1280px)

Exemple:

```
@media only screen and (min-width : 481px) and (max-width : 989px)
{
        /* Styles pour tablettes */
}

@media only screen and (max-width : 480px)
{
/* Styles pour mobiles */
}
```

La bonne pratique consiste à déterminer les points de bascules quand c'est nécessaire, en prenant en compte le contenu lors du redimensionnement. On peut lier une feuille CSS à un média (et ce depuis CSS2)

Exemple:

en html

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

en css

```
@media only screen
{
        […]code CSS[…]
}
```

Type de médias

- screen : écrans
- speech : synthèse vocale
- handheld : mobiles (souvent ignoré)
- braille : plages brailles
- print : impression
- embossed : imprimantes braille
- tv : télévision (peu compatible)
- projection : projecteurs
- all : tous
- tty : terminal, polices à pas fixe

Avec la norme CSS3 on peut **prendre en compte les caractéristiques du média**. Notamment du viewport en utilisant de nouveaux opérateurs: **min- et max.** Il est possible d'utiliser min- et max- devant la plupart des fonctionnalités pour spécifier des valeur minimum et maximum.

- And : et
- , : ou
- not : non (contraire de)
- only : pour la compatibilité

en html

```
<link rel="stylesheet" type="text/css" href="print.css" media="print and (color)">
```

en css

```
@media [not | only] screen [and] (expression){….}
@media print, screen and (not monochrome){ [...]code CSS[…] }
@media screen and (min-width:480px) { [...]code CSS[…] }
@media all and (orientation:landscape) { [...]code CSS[…] }
@media screen and (monochrome) { [...]code CSS[…] }
```

Voici un petit tableau récapitulatif des nouvelles propriétés CSS3 avec lesquelles vous pouvez travailler lorsque vous utilisez des mediaqueries :

Mediaqueries : fonctionnalités

Fonctionnalités	
color	… and (color) : périphérique couleur
color-index	… (min-color-index:256) : gérant au moins 256 couleurs indexées
aspect-ratio	… (aspect-ratio:16/9) ou 1/1, etc. : ratio de la zone d'affichage
height	… and (height : 800px) : hauteur de la zone d'affichage
width	… and (width : 800px) : largeur de la zone d'affichage
device-aspect-ratio	… and (device-aspect-ratio: 16/9) : ratio du périphérique
device-height	… and (max-device-height: 800px) : hauteur du périphérique
device-width	… and (max-device-width: 800px) : largeur du périphérique
grid	Détecte les grilles (terminaux, tels simples). Peu compatible.
monochrome	… and (monochrome) : périphériques noirs et blancs
orientation	and (orientation: portrait) : detecte le mode portrait. (ou landscape)
resolution	and (min-resolution: 300dpi) : détecte la résolution
scan	and (scan: progressive) : balayage (ou interlance). Peu compatible.

Mediaqueries en JavaScript

On peut très bien utiliser les mediaqueries avec du javascript, c'est d'ailleurs l'une des méthodes utilisées par beaucoup de librairies et framework pour nous permettre d'utiliser les mediaqueries sur des navigateurs anciens qui ne les supportent pas.

On peut utiliser la propriété matchMedia de l'objet window:

```
if (window.matchMedia("(min-width: 600px)").matches)
{
	console.log("600px ou plus");
}
else
{
	console.log("moins de 600px");
}
```

On la propriété clientWidth de l'objet document.body:

```
var sw = document.body.clientWidth;

if (sw < 480)
{
	console.log("moins de 480px");
}
if (sw >= 480)
{
	console.log("au moins 480px");
}
```

Voici un petit tableau récapitulatif du support des mediaqueries en fonction des versions de navigateurs:

IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
		4.0									
		5.0									
	2.0	6.0									
	3.0	7.0									
	3.5	8.0									
	3.6	9.0									
	4.0	10.0									
	5.0	11.0									
	6.0	12.0									
	7.0	13.0									
	8.0	14.0									
	9.0	15.0									
	10.0	16.0		9.0							
	11.0	17.0		9.5-9.6							
	12.0	18.0		10.0-10.1							
	13.0	19.0		10.5							
	14.0	20.0		10.6			2.1				
5.5	15.0	21.0	3.1	11.0			2.2		10.0		
6.0	16.0	22.0	3.2	11.1	3.2		2.3		11.0		
7.0	17.0	23.0	4.0	11.5	4.0-4.1		3.0		11.1		
8.0	18.0	24.0	5.0	11.6	4.2-4.3		4.0		11.5		
9.0	19.0	25.0	5.1	12.0	5.0-5.1		4.1	7.0	12.0		
10.0	20.0	26.0	6.0	12.1	6.0	5.0-7.0	4.2	10.0	12.1	25.0	19.0
	21.0	27.0									
	22.0	28.0									

Principe de grille flexible, fluide

Il existe beaucoup de grilles différentes avec autant de cas concrets d'utilisations. Nous allons parler ici uniquement des grilles "standards" dédiées au média web. Bien entendu, vous pouvez créer votre propre grille : mais ce n'est pas si simple qu'il n'y paraît - évitons de réinventer la roue.

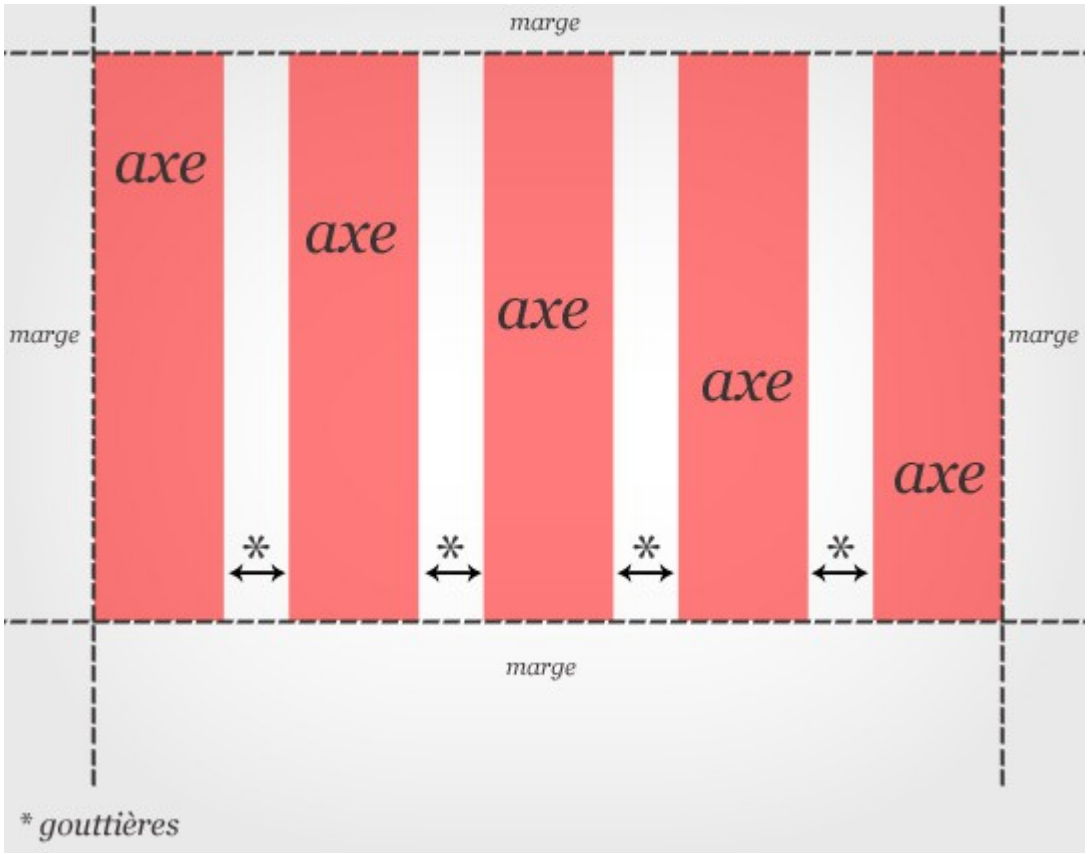
La majorité des grilles disposent de marges latérales, afin que l'œil puisse bénéficier d'un confort de lecture en ne "tapant" pas contre les bords de la fenêtre du navigateur

Anatomie d'une grille:

Une grille est constituée de repères (axes) verticaux et/ou horizontaux, séparés par des gouttières permettant de structurer le contenu. Il ne faut pas oublier les marges externes (pour les éventuels bords de la fenêtre) et internes (de chaque axe).

Elle sert d'armature pour organiser la page et son contenu. C'est une aide autant pour le designer que pour le visiteur. Cela permet d'atteindre un équilibre harmonieux entre l'espacement et l'organisation des différents blocs.

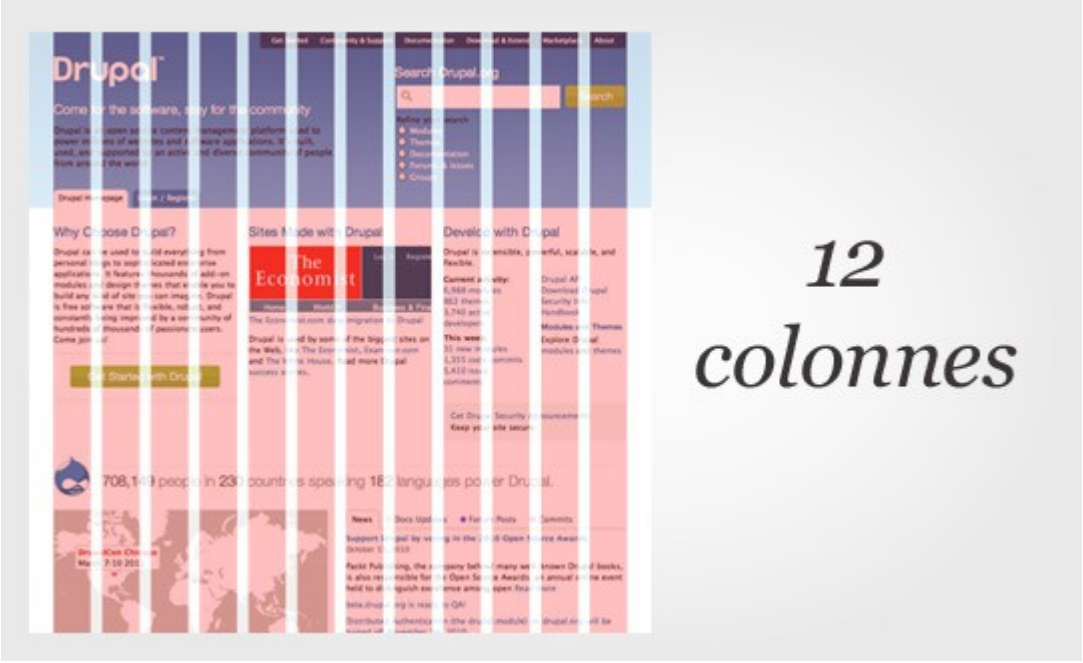
On obtient alors une page plus lisible et plus homogène. Sans grille, un site avec beaucoup de contenu risque d'être moins agréable à visiter.

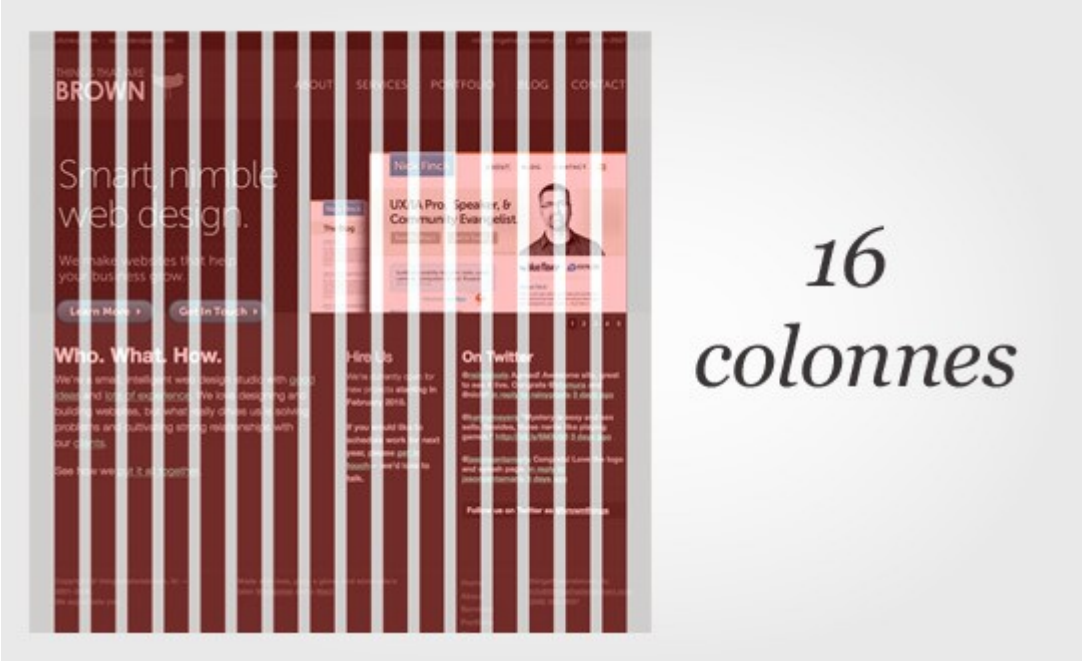


.

Les grilles verticales

On peut choisir le nombre de colonnes à insérer dans une largeur fixe. Par conséquence : moins il y a de colonnes plus elles sont larges.

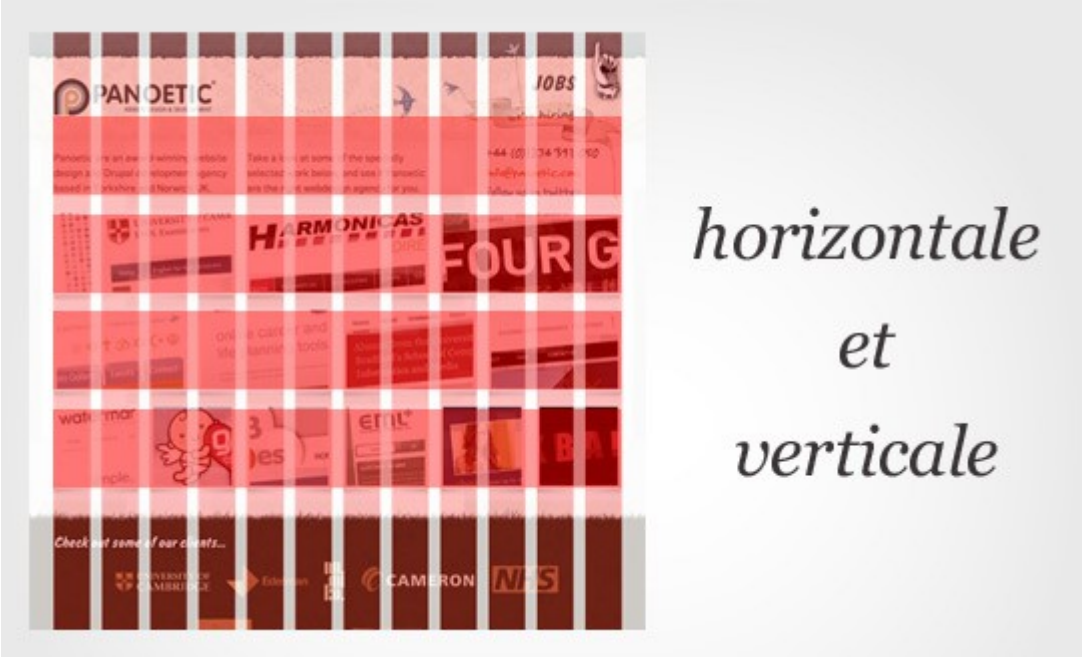




Ces colonnes permettent de donner un rythme de division dans le sens horizontal (de gauche à droite). La plupart du temps cela suffit pour créer un design.

Les grilles verticales et horizontales

Ajouter des divisions horizontales permet de donner des repères dans le sens vertical (de haut en bas).



Comment choisir sa grille ?

Il n'y a pas vraiment de règle type : tout dépend du projet. Il est donc très important de bien préparer son projet en aval pour, entre autre, bien cerner le contenu du site. A ce propos, un article sur Alsacréations pourra peut être vous intéresser : Mockups & Rough : gagnez du temps !

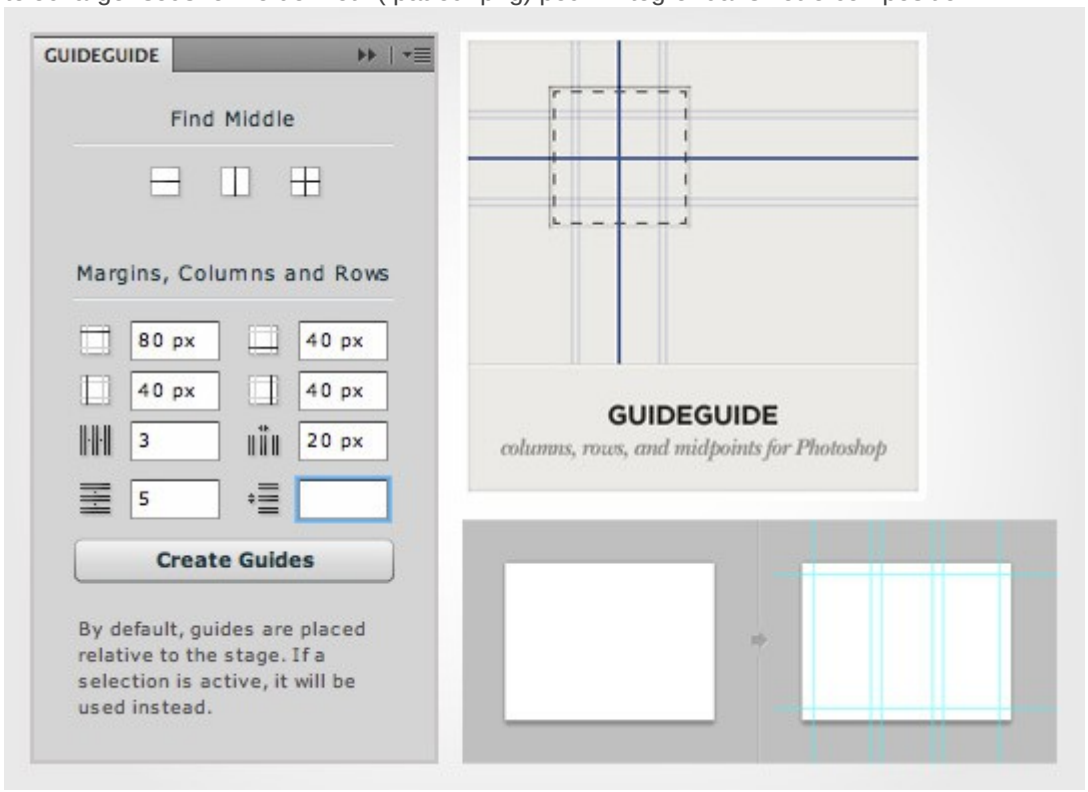
Votre grille devra donc être compatible avec les longueurs de texte, le type de site (institutionnel, portfolio, blog/magazine...), etc. On peut penser que plus il y a de contenu, plus nous aurons besoin de colonnes dans la grille. Mais ce n'est pas une règle absolue, il s'agit toujours de faire un compromis entre quantité d'information présentée à l'écran et lisibilité.

Où trouver des grilles ?

De nombreux sites proposent de télécharger des modèles de grilles. Cependant, certains sites ne proposent que des Frameworks CSS qui ne sont pas toujours utilisables directement dans un logiciel de design.

"960 gs" est très connu, en plus de proposer un framework CSS, il est possible de télécharger des templates pour nombres de logiciels (Photoshop, Fireworks, GIMP...). Le template se présente sous la forme d'un calque : il suffit de le placer en première position dans votre palette et le tour est joué.

"Modular Grid Pattern", permet de créer à la volée une grille directement sur le site. Vous pourrez ensuite la télécharger sous forme de motif (.pat ou .png) pour l'intégrer dans votre composition.

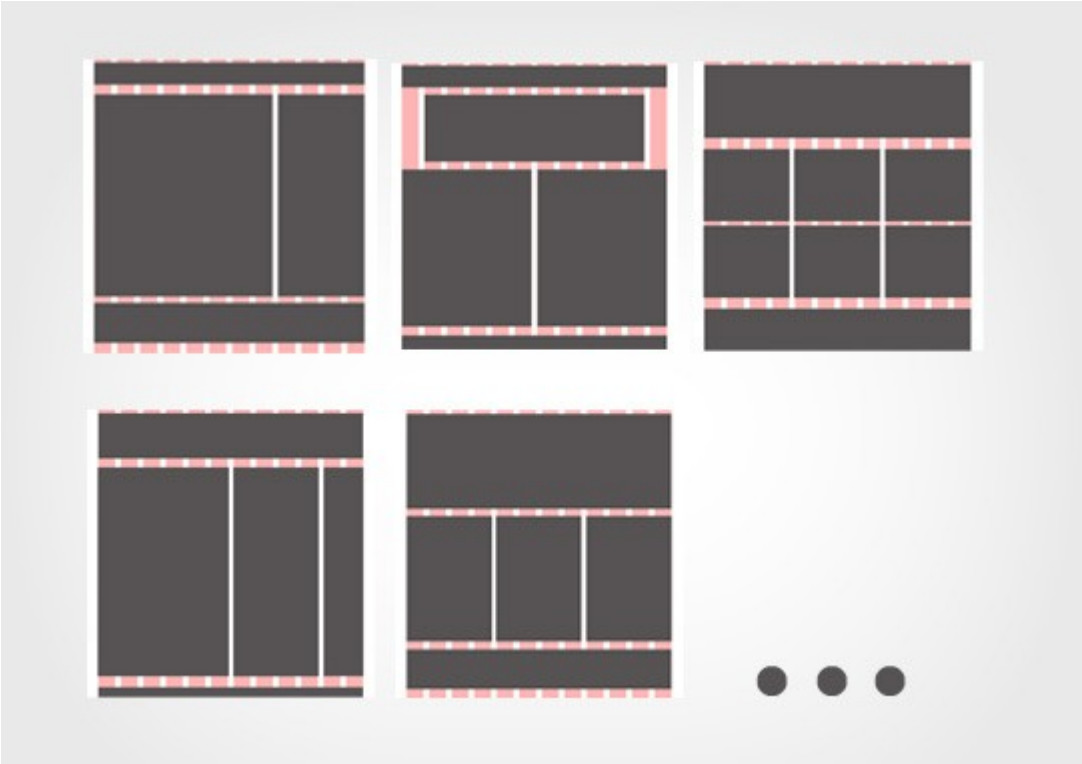


"GuideGuide" est une extension Photoshop (à partir de CS4) permettant de générer des repères symbolisant une grille. Pour information, c'est encore en version bêta.

Agencement des grilles (blocs)

Après avoir choisi sa grille, il faut maintenant la "remplir" en disposant les différents blocs qui vont accueillir le contenu (images, textes, vidéos, diaporama...).

Énormément de combinaisons existent, voici quelques unes "standards" :



Concevoir un site web en suivant le principe de la grille est un sujet vaste et très intéressant. Le connaître vous permettra d'être plus à l'aise dans vos réalisations (autant à destination de l'impression que du web). N'hésitez pas à faire un tour dans le "temple de la grille", nommé "the grid system". On y trouve des ouvrages à lire, des articles, des outils... bref ce sont des *aficionados des grilles* ! N'hésitez pas non plus à jeter un coup d'œil à l'excellente conférence d'Anne-Sophie Fradier (Paris Web 2010).

Les grilles ne sont en aucun cas des freins à la création graphique : bien au contraire ! On part d'un cadre sain et efficace : c'est rassurant et il ne demande qu'à être enfreint (à condition qu'on le fasse bien). C'est une aide précieuse pour le designer (il peut dormir avec). Peu importe votre type de projet (beaucoup de contenu ou très graphique) partez toujours d'une grille : vous serez libre par la suite de prendre des libertés, mais le bon départ sera quasiment assuré.

Les points de ruptures:

Il est important de noter que, du fait que les grilles divisent naturellement l'espace disponible en "n" portions de mêmes tailles, il est nécessaire de construire a minima, une règle de séparation de cet espace pour chaque type de device clairement identifié.

Nous avons déjà parlé des points de ruptures lors du chapitre sur les media queries, nous pouvons réutiliser ce système afin de définir une taille précise pour les colonnes / blocs de nos grilles.

Une "bonne solution" pourrait consister en la division de l'espace à l'aide de l'unité de mesure % et non une unité fixe comme le pixel, toutefois cette solution a ses avantages et ses inconvénients que nous détaillerons lors de la partie pratique.

Passons maintenant à un sujet important, les composants graphiques et le modèle flexbox.

Les composants graphiques:

Principe de la typographie responsive

Il existe deux façons principales de mettre en place une typographie responsive. La première est le redimensionnement de la police. Cela signifie que la typographie est redimensionnée en fonction de la taille de l'écran mais qu'elle peut également être redimensionnée par l'utilisateur.

La deuxième est d'optimiser la longueur des lignes pour maintenir la lisibilité. Cela signifie que pour certains écrans il est plus logique de réduire la zone de contenu et donc la longueur des lignes même si le contenu peut être plus large.

Nous allons nous concentrer uniquement sur la première solution qui est beaucoup plus facile à mettre en oeuvre

Redimensionner la typographie en utilisant les unités rems

La plupart des concepteurs utilisent soit les pixel (px) soit les ems comme unités pour le dimensionnement de leur typographie. Utiliser les ems est une meilleures option que les pixels, car ils permettent aux utilisateurs de redimensionner la typographie dans leur navigateur. Mais la taille en em est relative à la taille de police des éléments parents et est plus compliquée à utiliser que la taille en pixel et plus compliquée encore dans la mise en place d'un responsive design.

L'unité rem est la meilleure alternative à em. Son fonctionnement est presque identique à une différence près essentielle : les unités rems sont relatives à l'élément et non aux éléments parents. Cela permet de grandement faciliter la maintenance.

Les unités rems sont maintenant supportées par la plupart des navigateurs modernes comme Firefox depuis la version 3.6, Opera depuis la version 11.6, Internet Explorer depuis la version 9… Vous pouvez mettre en place des solutions de repli pour les anciens navigateurs mais les rems sont assez bien supportés pour les utiliser maintenant.
Maintenant que vous allez utiliser les rems pour dimensionner votre typographie, assurez vous de faire un reset sur l'élément html et non sur l'élément body. Cela devrait ressembler à :

en css

```
html
{
	font-size:100%;
}
```

Maintenant votre unité rem sera fonction de la taille de police par défaut du périphérique.

Ensuite vous aurez besoin de spécifier la taille de police pour chaque taille d'écran. Je recommande d'expérimenter les mêmes tailles de polices sur des appareils différents pour voir ce qui semble le mieux. Cela dépend des polices de caractères que vous avez choisies ainsi que de votre conception globale.

Vous aurez probablement envie de spécifier de multiples tailles de police basées sur différentes tailles d'écran, ce qui est assez simple à faire. Par exemple, votre CSS pourrait ressembler à ceci:

```
@media (max-width: 640px) {
 body {
  font-size:1.2rem;
 }
}
@media (min-width: 640px) {
 body {
  font-size:1rem;
 }
}
@media (min-width:960px) {
 body {
  font-size:1.2rem;
 }
}
@media (min-width:1100px) {
 body {
  font-size:1.5rem;
 }
}
```

Je recommande fortement d'utiliser un outil comme Web Font Specimen pour voir de quoi a l'air votre typographie.

Nous allons maintenant parler du nouveau modèle de boîte flexible de css3, j'ai nommé le **modèle flexbox**

Vous connaissez certainement le modèle de boîte classique en CSS et ses dispositions de type "block" ou "inline", sachez que Flexbox CSS3 a été conçu pour étendre ce périmètre en introduisant un nouveau modèle de boîte distinct, que l'on appellera **"le Modèle de boîte flexible"**.

Au sein de ce schéma, on ne raisonne plus en "block" ou "inline", ni même en float ou autres types de boîtes "classiques" CSS, mais en "Modèle de boîte flexible", dont les quatre possibilités principales sont :

1. Distribution des éléments horizontale ou verticale, avec passage à la ligne autorisé ou non,
2. Alignements et centrages horizontaux et verticaux, justifiés, répartis,
3. Réorganisation des éléments indépendamment de l'ordre du flux (DOM),
4. Gestion des espaces disponibles (fluidité).

Flexbox (le modèle de boîte flexible) se fonde schématiquement sur une architecture de ce type :

•Un "flex-container" permettant de créer un contexte général d'affichage,
•Un ou plusieurs "flex-item" qui ne sont rien d'autre que les enfants directs du conteneur, quels qu'ils soient.

Le "flex-container", qui définit le contexte global de modèle de boîte flexible, est tout simplement n'importe quel élément HTML doté de la déclaration display: flex; oudisplay: inline-flex;.
Ses enfants deviennent alors automatiquement (inutile de leur déclarer quoi que ce soit) des éléments de type "flex-item" :

```
.container {
    display: flex;
}
```

Un élément "flex-item" n'est plus considéré comme un "bloc" ou un "inline" classique (d'ailleurs les valeurs de display autre que none, et même certaines propriétés telles que float n'ont plus d'effet sur lui).

Distribution et axe principal

La distribution, c'est à dire le sens d'affichage horizontal ou vertical des éléments "flex-items" est définie par la propriété flex-direction dont les valeurs peuvent être :

- row (distribution horizontale, valeur par défaut)
- row-reverse (distribution horizontale inversée)
- column (distribution verticale)
- column-reverse (distribution verticale inversée)

Cette propriété s'applique au "flex-container" et détermine l'axe principal du modèle de boîte flexible.

```
.container {
    display: flex;
    flex-direction: column;
}
```

La propriété flex-wrap définit si le contenu sera distribué sur une seule ligne (ou colonne selon l'axe principal) ou sur plusieurs lignes. En clair, si les "flex-items" ont le droit de passer à la ligne ou non.

Les valeurs de flex-wrap sont :

- nowrap (les éléments ne passent pas à la ligne, valeur par défaut)
- wrap (les éléments passent à la ligne dans le sens de lecture)
- wrap-reverse (les éléments passent à la ligne dans le sens inverse)

```
.container {
    display: flex;
    flex-wrap: wrap-reverse;
}
```

À noter qu'il existe une propriété raccourcie flex-flow qui regroupe flex-direction etflex-wrap .

```
/* affichage en ligne et passage à la ligne autorisé */
.container {
    flex-flow: row wrap;
}
```

Alignements

Flexbox propose de gérer très finement les alignements et centrages, en différenciant les deux axes d'affichage de cette manière :

-L'alignement dans l'axe principal est traité via la propriété justify-content

- L'alignement dans l'axe secondaire est géré avec align-items

Ces deux propriétés s'appliquent au "flex-container".

Axe principal : justify-content

Les alignements dans l'axe de lecture principal sont définis à l'aide de la propriétéjustify-content, dont les valeurs possibles sont :

- flex-start (éléments positionnés au début du sens de lecture, valeur par défaut)
- flex-end (éléments positionnés à la fin)
- center (position centrale)
- space-between (répartition "justifiée")
- space-around (variante de répartition "justifiée")

```
/* éléments positionnés en bas du conteneur */
.container {
    flex-direction: column;
    justify-content: flex-end;
}
```


```css
.parent {
   display: flex;
   flex-direction: column;
   justify-content: flex-end;
}
```

Axe secondaire: align-items

Dans l'axe secondaire, les alignements sont régis via la propriété align-items, dont les valeurs sont :

- flex-start (au début)
- flex-end (à la fin)
- center (au centre)
- baseline (généralement identique à flex-start)
- stretch (étirés dans l'espace disponible, valeur par défaut)

```css
/* éléments étirés (valeur par défaut) */
.container {
    flex-direction: column;
    align-items: stretch;
}
```


```css
.parent {
   display: flex;
   flex-direction: column;
   align-items: stretch;
}
```

Traiter les cas particuliers : align-self

La propriété align-self , permet de distinguer l'alignement d'un "flex-item" de ses frères. Les valeurs de cette propriété sont identiques à celles de align-items.

```
/* seul le paragraphe sera à droite */
.container {
    align-items: stretch;
}
p {
    align-self: flex-end;
}
```


```
.parent {
   display: flex;
   flex-direction: column;
   align-items: stretch;
}
.oignon {
   align-self: flex-end;
}
```

Propriété margin

La propriété margin lorsqu'elle est affectée à un "flex-item" ouvre de nouvelles perspectives, notamment dans l'axe vertical puisque Flexbox n'est plus lié à un sens de lecture en particulier.

En clair, il devient possible de positionner un élément en bas de son conteneur à l'aide d'un margin-top: auto, ou mieux : centrer à la fois horizontalement et verticalement via un simple margin: auto.

```
/* paragraphe centré horizontalement et verticalement */
.container {
    display: flex;
}
.container > p {
    margin: auto;
}
```

Ordonnancement

L'une des fonctionnalités les plus avant-gardistes du modèle d'affichage Flexbox est de pouvoir réordonner à sa guise chacun des éléments indépendamment grâce à la propriété order.

Les valeurs de order agissent telles des pondérations : les éléments dont la valeur est la plus forte se trouveront en bas de la pile.

La propriété order s'applique aux "flex-items" et sa valeur initiale est 0.

```
/* le premier de la liste s'affichera en bas de pile */
li:first-of-type {
    order: 1;
}
```


```
.oignon {
   order: 1;
}
```


```
.picon-biere {
   order: -1;
}


.oignon {
   order: 1;
}
.salade {
   order: 2;
}
```

Flexibilité

Cela ne devrait étonner personne, la notion de flexibilité constitue le fondement du module de positionnement Flexbox, et c'est là qu'intervient l'indispensable propriétéflex.

La propriété flex est un raccourci de trois propriétés, flex-grow, flex-shrink et flex-basis, qui s'appliquent aux "flex-items" et dont les fonctionnalités sont:

- flex-grow : capacité pour un élément à s'étirer dans l'espace restant,
- flex-shrink : capacité pour un élément à se contracter si nécessaire,
- flex-basis : taille initiale de l'élément avant que l'espace restant ne soit distribué.

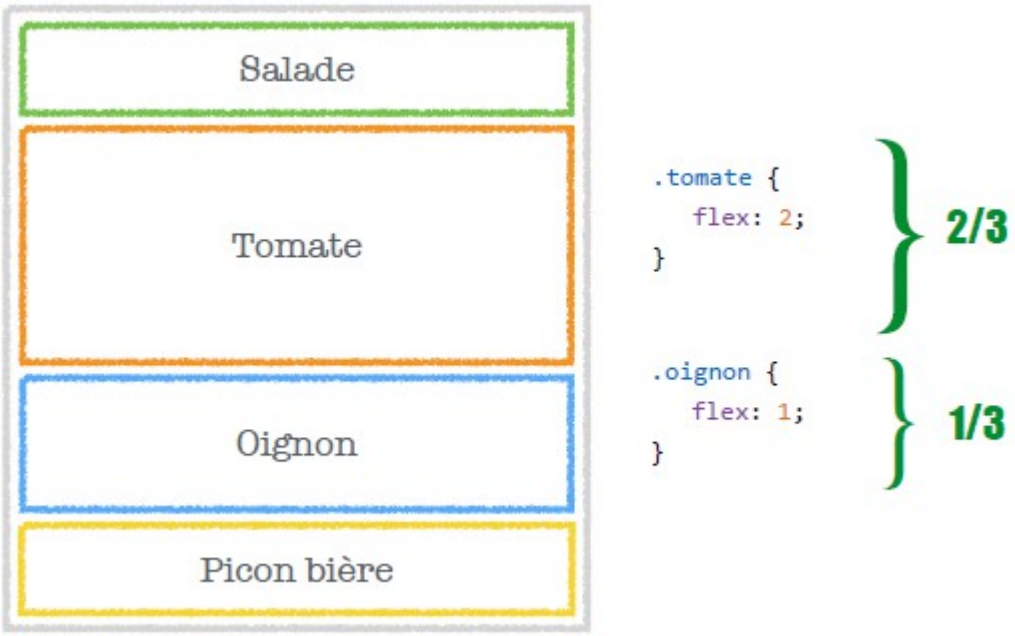
Par défaut, les valeurs de ces propriétés sont : flex-grow: 0, flex-shrink: 1 et flex-basis: auto.
En clair, les flex-items n'occupent initialement que la taille minimale de leur contenu.

Pour rendre un élément flexible, il suffit de lui attribuer une valeur de flex-grow (ou flexen raccourci) supérieure à zéro.

Cet élément occupera alors l'espace restant au sein de son conteneur :

```
/* .salade occupera l'espace restant */
.salade {
    flex: 1;
}
```

Plusieurs éléments peuvent être rendus flexibles et se répartir l'espace restant. L'espace disponible est alors tout simplement distribué entre les éléments flexibles.



Méthodes et frameworks

Les navigateurs anciens et surtout Internet Explorer **interprètent mal le CSS3** et les balises HTML 5. Afin de "mettre à plat" les différents styles des divers navigateurs on peut utiliser le fichier **reset.css de Eric Meyer** disponible à l'adresse: http://meyerweb.com/eric/tools/css/reset/

Pour le **support des balises HTML5 non supportées** (de plus en plus rare) on peut inclure la librairie javascript **html5Shiv** (balises) disponible sur internet.

Mais la solution la **plus complète et la plus simple**, à condition de passer un peu de temps à s'auto former, est l'utilisation d'un **framework**.

Un framework est une suite de fonctionnalités logicielles prédéfinies, qui comme leur nom l'indique, **va vous donner une structure prédéfinie** à respecter ainsi que toute une suite d'outils respectant cette structure et qui ont pour but de vous faciliter la vie (donc de **prendre en charge tout ce qui est fastidieux** ou presque)

Il en existe des dizaines et nous n'allons pas détailler leur fonctionnement dans ce cours, nous allons nous contenter d'en lister quelques-uns et je vous montrerai un peu comment vous y prendre pour attaquer les plus célèbres, **bootstrap et Modernizr.**