Introduction au responsive design

Rédigé par: Jean-Luc Colson			
Date première rédaction: Janvier 2020			
SUIVI DES MODIFICATIONS A LA FICHE			
Série	Date	Page(s) modifiée(s)	Raison

Contents

Introduction au responsive design

Dans cette dernière partie, nous nous attaquons à une notion clef du développement d'un site Internet : le responsive design ou « design adaptable ».

Nous allons voir ici les principaux défis apportés par la multiplication des appareils pouvant aller sur le web et comment y répondre efficacement.

Comprendre les défis apportés par la multiplication des écrans

Il y a quelques années encore, on ne pouvait accéder à Internet et au web que grâce à des ordinateurs de bureau.

Avec la miniaturisation des composants et les capacités de connexion de plus en plus puissante, cependant, de nouveaux appareils connectés ont vu le jour et les sites web peuvent maintenant être consultés à partir de nombreux appareils différents : ordinateurs de bureau, ordinateurs portables, tablettes, smartphones, montres connectées, etc.

Cela a multiplié les défis en termes d'ergonomie et de design pour les concepteurs de site : en effet, comment faire en sorte qu'un site Internet s'affiche correctement à la fois sur un écran de bureau et sur un smartphone ? Doit-on créer deux sites différents ? Deux versions d'un même site ? Dans ce cas-là, est-il préférable de retirer des informations pour rendre la version mobile plus légère ou peut-on se « contenter » de réarranger le contenu ?

Une définition rapide du responsive design

Lorsqu'on parle de « responsive design » ou de « design adaptable » en français, on fait référence à l'idée selon laquelle un site web devrait s'afficher aussi bien sur un écran de PC que sur un écran de smartphone ou sur n'importe quel type d'appareil.

Aujourd'hui, nous pouvons utiliser principalement trois méthodes pour répondre aux défis amenés par les différentes tailles d'écran. On peut :

- Créer une application dédiée pour les mobiles ;
- Créer une « copie mobile » de notre site en utilisant l'initiative AMP (« Accelerated Mobile Pages ») de Google ;
- Utiliser les Media Queries ou requêtes media.

L'idée de base du responsive design est qu'un site web devrait présenter les mêmes informations quel que soit l'appareil qui l'affiche, en les réarrangeant pour conserver la meilleure ergonomie possible. Nous allons pouvoir achever cela en utilisant les Media Queries qui seront donc la notion centrale de cette partie.

Définition rapide des Media Queries

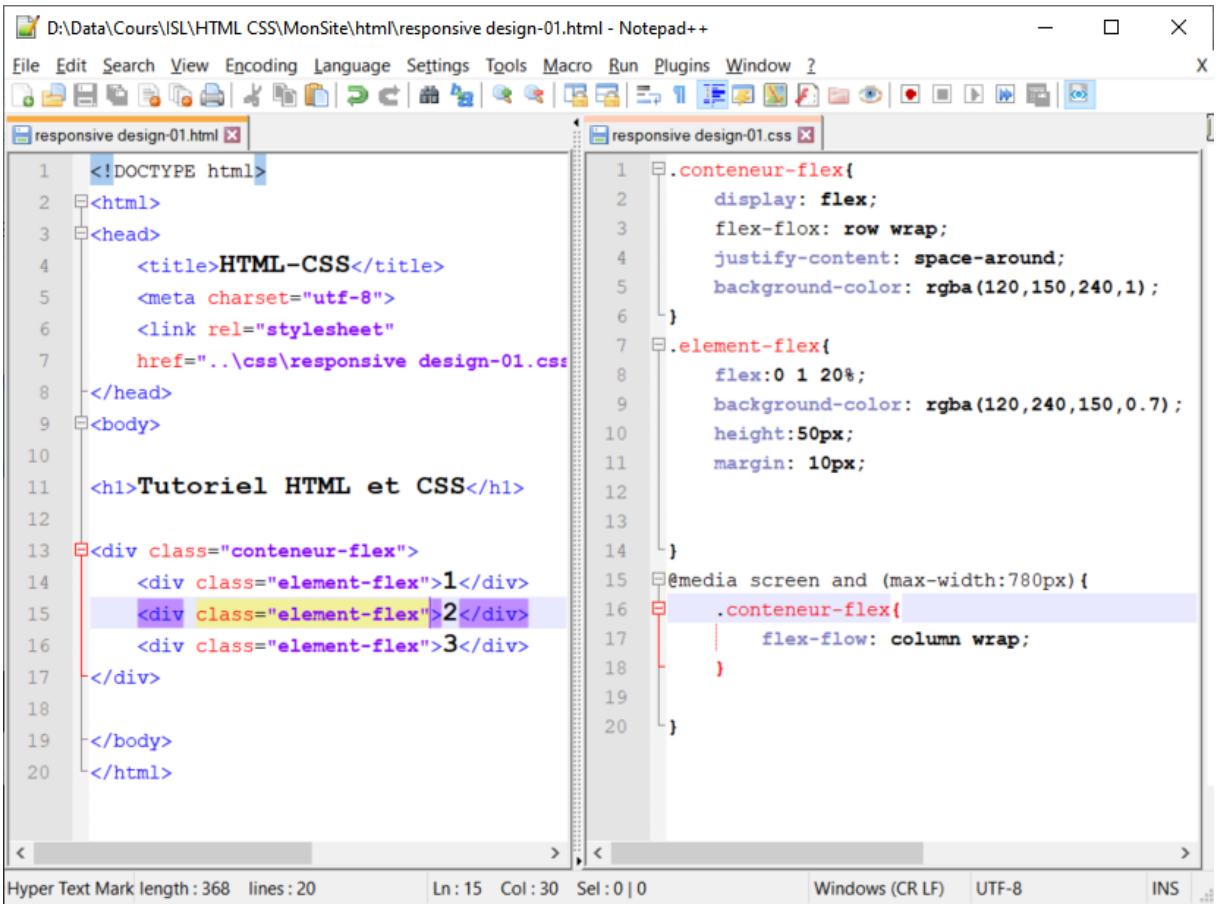
Les Media Queries vont nous permettre d'appliquer certaines règles CSS de manière conditionnelle. Par exemple, on va pouvoir définir une largeur pour un élément pour certaines tailles d'écrans et une largeur différente pour le même élément pour d'autres tailles d'écran.

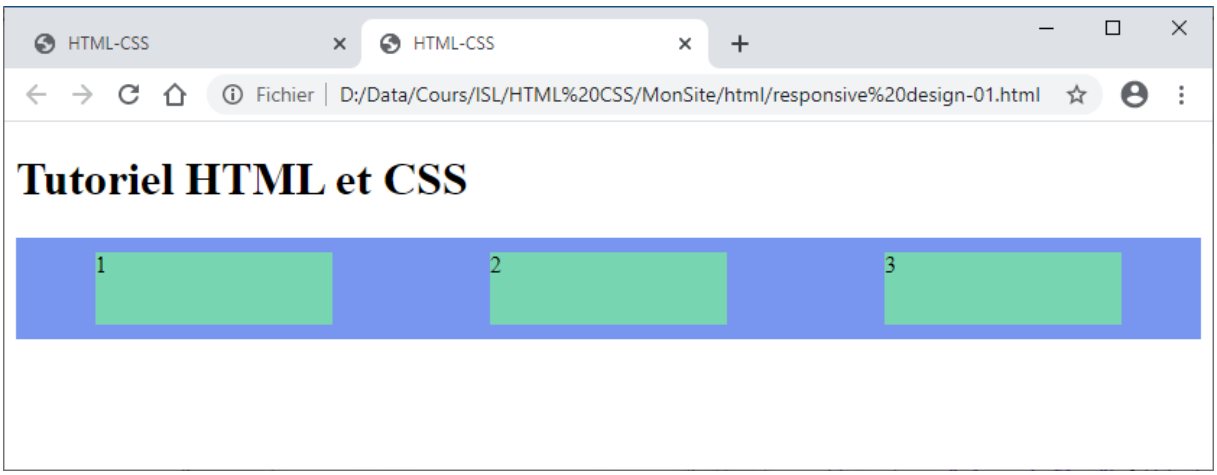
Cela va nous permettre d'afficher des pages avec des organisations différentes selon la taille de l'écran d'un visiteur. Attention ici à ne pas confondre les Media Queries et les valeurs de taille relatives et en % ou le flexbox.

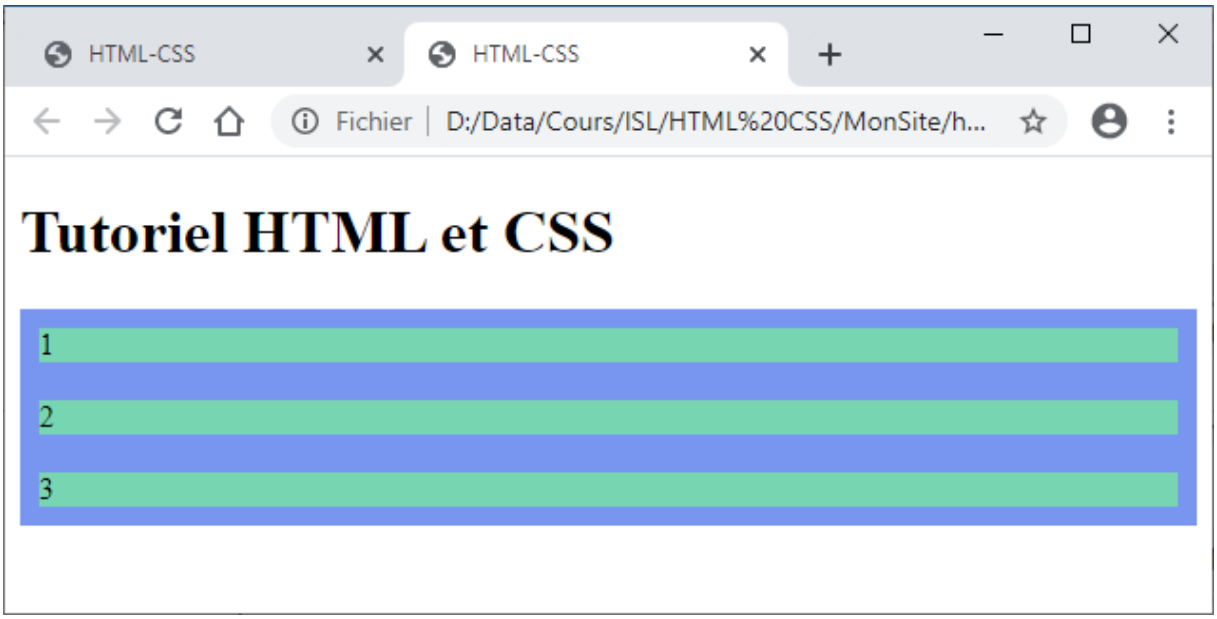
Les valeurs en % et le flexbox vont pouvoir permettre aux éléments de grandir ou de rétrécir selon la taille d'un écran mais selon la même règle CSS.

Avec les Media Queries, nous allons pouvoir appliquer des règles CSS totalement différentes selon les tailles d'écrans. Notez par ailleurs qu'on va tout à fait pouvoir utiliser le flexbox, etc. dans nos Media Queries.

Regardez l'exemple ci-dessous pour bien comprendre :







Ici, on commence par définir un conteneur flexible qui va contenir trois éléments flexibles.

On définit ensuite l'axe principal et la direction de notre conteneur avec flex-flow : row wrap. Notre axe principal va donc être l'axe horizontal par défaut et les éléments flexibles vont s'afficher les uns à côté des autres. Les éléments seront distribués régulièrement le long de leur axe principal avec justify-content: space-around.

On définit finalement une dimension de 20% pour nos éléments flexibles dans leur axe principal et on leur interdit de grossir automatiquement.

Ensuite, on va définir une règle media avec @media. Ici, notre règle va s'appliquer à tous les appareils disposant d'un écran (screen) de taille inférieure ou égale à 780px (max-width : 780px).

Dans cette règle media, nous allons pouvoir écrire autant de règles CSS que l'on souhaite. Ces règles CSS ne vont être appliqués que lorsqu'un utilisateur affiche la page avec un écran ou dans une fenêtre de taille inférieure à 780px.

Ici, on choisit de définir la direction de notre conteneur flexible avec flex-flow: column wrap pour tous les écrans de taille inférieure à 780px.

Notez qu'en cas de conflit (c'est-à-dire si une même propriété a déjà été définie en dehors d'une règle media ou dans une autre règle media), c'est la valeur de la propriété définie dans la règle la plus précise ou la plus restrictive qui va s'appliquer.

Dans le cas présent, nos éléments flexibles s'afficheront donc en ligne de manière générale et en colonne dès que la fenêtre fera moins de 780px. N'hésitez pas à modifier la taille de votre fenêtre pour bien voir le changement s'opérer.

Ce type de modification de structure ne serait pas possible en utilisant juste le flexbox ou des valeurs en % et ne va pouvoir être réalisé qu'avec les Media Queries.

La balise meta viewport

Avant d'étudier les Media Queries et la création de design responsives en profondeur, il est important de comprendre le rôle de la balise <meta name= "viewport">.

L'élément HTML meta

L'élément HTML meta est utilisé pour définir des métadonnées pour un document HTML.

Une métadonnée est une donnée qui ne va pas être affiché sur la page mais qui va pouvoir servir aux différents robots pour comprendre et afficher la page.

On va pouvoir ajouter différents attributs à l'élément meta qui vont nous servir à spécifier différents types de données.

Les attributs les plus courants de l'élément meta vont être les attributs charset, name et content.

Nous connaissons déjà l'attribut charset qui sert à indiquer le jeu de caractères ou l'encodage de notre document. Cet attribut va notamment permettre aux navigateurs d'utiliser le bon jeu de caractères et d'afficher correctement les caractères de notre page et notamment les accents et autres caractères spéciaux comme les cédilles et etc.

L'attribut name va nous permettre d'indiquer le type de métadonnées que l'on souhaite passer. Cet attribut va aller de pair avec l'attribut content qui va lui nous permettre de passer une métadonnée en soi.

L'attribut name va notamment pouvoir prendre les valeurs suivantes :

- author : la valeur passée à content sera considérée comme étant le nom de l'auteur du document ;
- description : la valeur passée à content pourra être utilisée par les moteurs de recherche comme extrait pour décrire le sujet de notre page ;
- viewport : la valeur passée à content va nous permettre d'indiquer comment le navigateur doit afficher la page sur différents appareils.

Le viewport et les problèmes d'affichage sur mobile

Le viewport représente de manière schématique la partie visible d'une page web par un utilisateur ou la fenêtre active.

La taille de cette fenêtre va bien évidemment varier en fonction de la taille de l'écran de l'utilisateur et de l'appareil utilisé.

Le problème qui s'est posé avec les smartphones est que la taille du viewport, c'est-à-dire la taille de la fenêtre d'affichage des pages web va souvent être différente de la taille physique des écrans.

En effet, la plupart des smartphones utilisent un viewport plus grand que la taille réelle de leur écran afin d'éviter aux utilisateurs d'avoir à dézoomer dans le cas où ils consulteraient un site non optimisé pour une navigation sur mobile.

Ainsi, imaginons par exemple qu'un smartphone utilise un viewport de 980px de large. Le site va donc s'afficher dans cette fenêtre. Seulement, notre smartphone n'a une taille réelle d'écran que de 400px.

Bien évidemment, ici, le viewport ne va pas dépasser de l'écran physique mais va être recadré pour s'afficher dans l'écran. Ainsi, notre site va apparaitre comme dézoomé, puisque le niveau de zoom sera de 400 / 980 = 0,41x.

Le vrai problème ici est que tous les smartphones et les navigateurs mobiles n'utilisent pas les mêmes tailles de viewport ni les mêmes ratios. Ainsi, nos pages vont apparaitre plus ou moins dézoomées selon les appareils utilisés.

La balise meta name= "viewport" a été créée pour nous permettre de reprendre le contrôle du viewport et notamment de sa taille et de son échelle afin de proposer la meilleure version de notre site pour les différents appareils.

La balise meta name= »viewport »

La balise meta name= "viewport" va permettre de donner des instructions relatives à la taille et à l'échelle du viewport aux navigateurs mobiles afin que les différents éléments d'une page s'affichent au mieux.

Nous allons pouvoir passer plusieurs propriétés à l'attribut content.

Les propriétés width et height vont nous permettre de contrôler la taille du viewport dans lequel notre page doit s'afficher. On peut leur passer un nombre ou le mot clef device-width qui correspond à la taille de l'écran en pixels CSS à l'échelle 100%.

Ici, il me semble intéressant de définir ce qu'est « un pixel CSS ». Les pixels CSS correspondent à la surface utilisable de l'écran. Ce sont des pixels virtuels que l'appareil « pense » avoir. L'idée importante ici est qu'un pixel CSS n'est pas toujours égal à un pixel physique.

Les pixels physiques correspondent aux pixels réels qui composent un écran. C'est également ce qu'on appelle la définition d'un écran. Les écrans retina et haute définition possèdent généralement 4 fois plus de pixels réels que de pixels CSS.

La propriété user-scalable permet à l'utilisateur de zoomer dans la page (avec la valeur yes) ou, au contraire, lui interdit de la faire (avec la valeur no).

Cette propriété est souvent utilisée avec les propriétés minimum-scale et maximum-scale auxquelles on va pouvoir passer un nombre entre 0 et 10 et qui va représenter le niveau de dézoom ou de zoom que l'utilisateur est autorisé à faire.

Finalement, la propriété initial-scale permet de définir de niveau de zoom initial du viewport, c'est-à-dire son échelle. Nous allons également pouvoir lui passer un nombre entre 0 et 10.

Définir le meta viewport : quelles valeurs choisir ?

Si vous avez suivi jusque-là, vous devriez avoir compris qu'il est essentiel de définir une balise meta name= "viewport" avec des propriétés et des valeurs adaptées afin que les différents navigateurs mobiles ne définissent pas eux-mêmes leur propre viewport et leur niveau de zoom de notre page.

Généralement, nous définirons une largeur de viewport égale à la largeur de l'appareil dans le viewport ainsi qu'un niveau de zoom initial égal à 1 et interdirons les utilisateurs de zoomer ou de dézoomer. Nous allons donc utiliser la balise suivante :

```
<head>
    <title>Cours HTML et CSS</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
</head>
```

Utiliser les media queries pour créer un design responsive

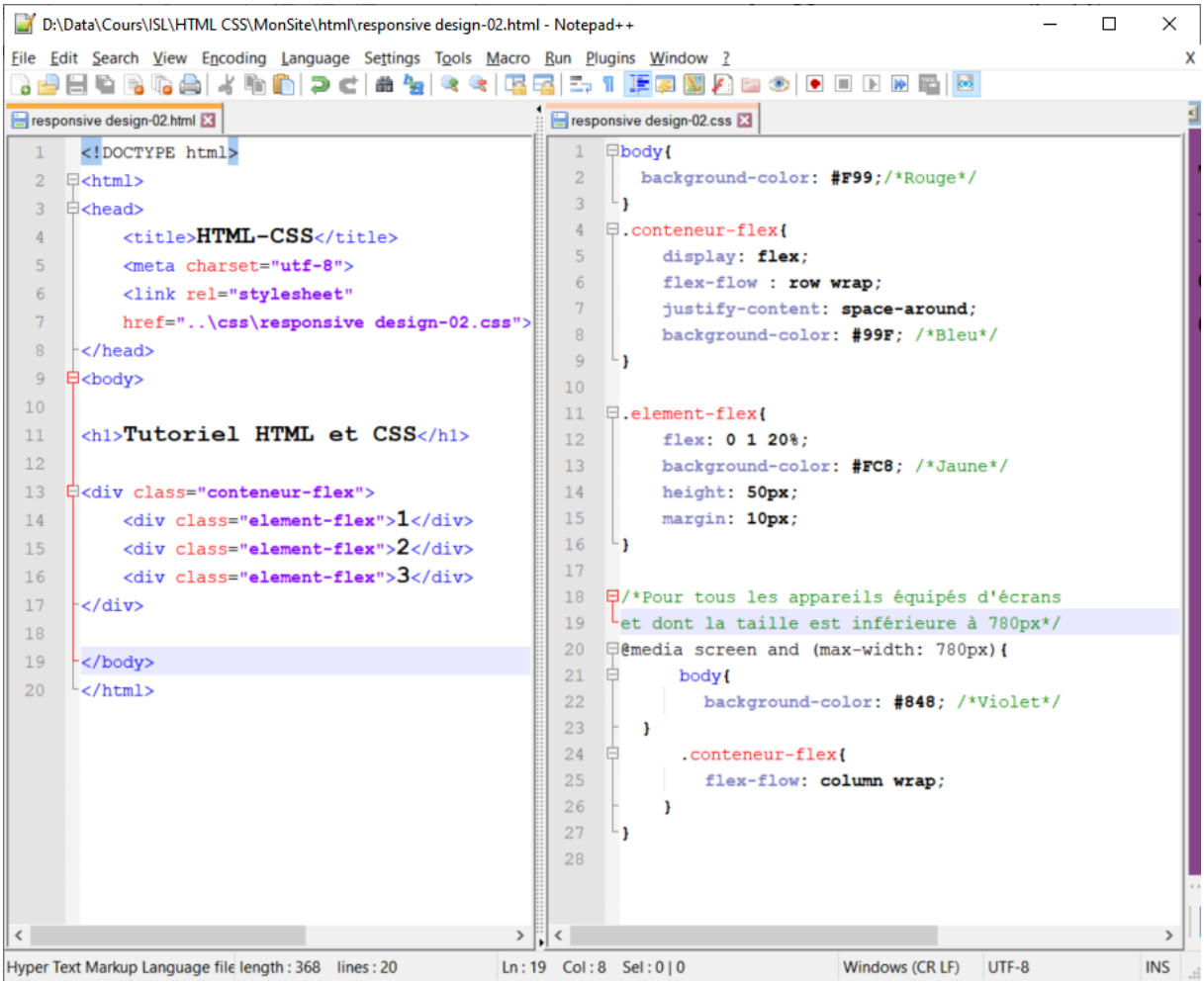
Dans cette nouvelle leçon, nous allons découvrir dans le détail ce que sont les requêtes media ou Media Queries, comprendre comment elles fonctionnent et apprendre à les utiliser.

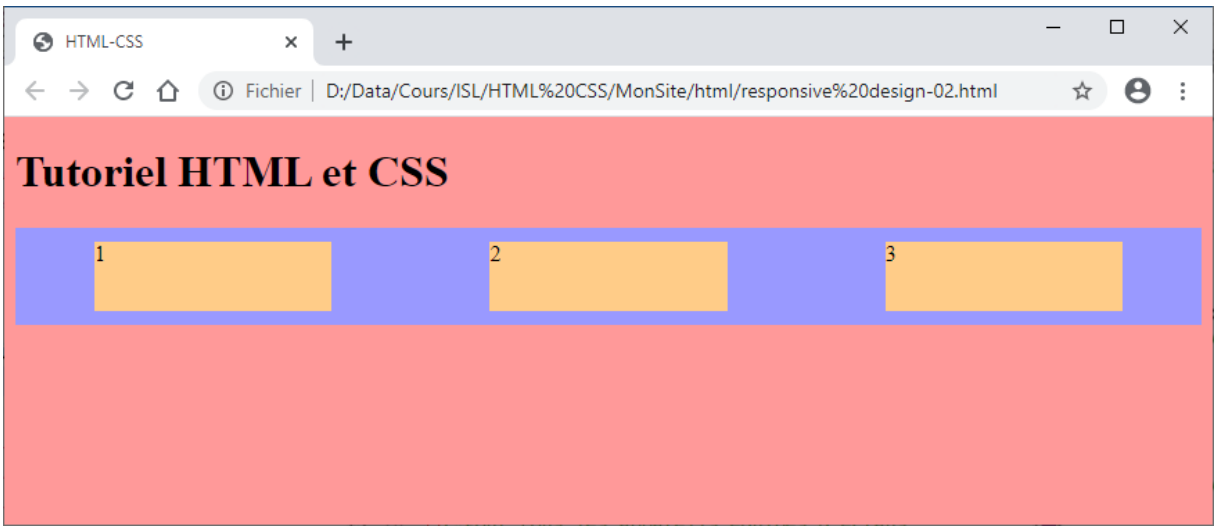
Définition des Media Queries

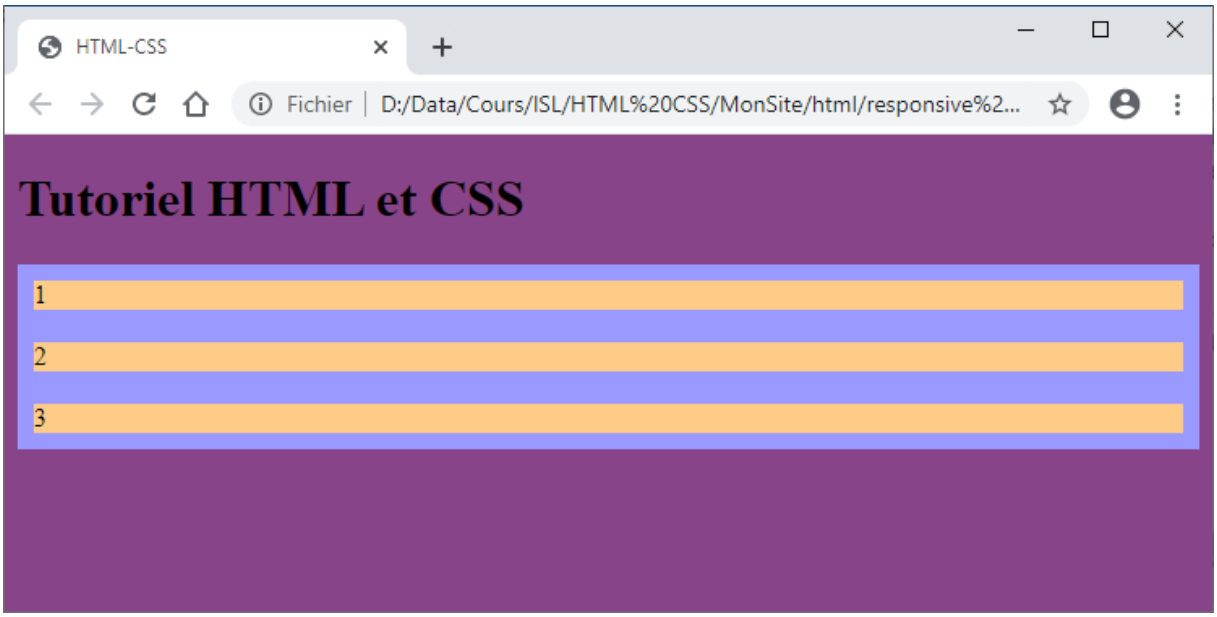
Les Media Queries correspondent à des styles CSS conditionnels. Les Media Queries se basent sur la règle CSS @media qui va nous permettre de définir différents styles CSS pour différents types d'appareils media et selon différents critères.

Nous allons ainsi, grâce aux Media Queries, pouvoir présenter le même contenu HTML de différentes façons en fonction de l'appareil utilisé par nos visiteurs pour accéder à nos pages.

On va ainsi par exemple déjà pouvoir changer la disposition des éléments ou la couleur de fond de nos pages en fonction de la taille de l'écran d'un utilisateur :







Le fonctionnement des Media Queries (syntaxe)

Dans notre règle @media, nous allons pouvoir placer deux types de conditions ou de contraintes : une condition sur le media utilisé pour afficher la page et une condition sur les caractéristiques du media.

A l'origine, on pensait que créer une condition sur le type de media serait suffisante et il n'y avait pas de conditions sur les caractéristiques des media.

Cependant, au fil du temps nous nous sommes rendu compte qu'il était beaucoup plus simple et plus logique d'ajouter des conditions sur les caractéristiques d'un media plutôt que sur son type.

Ainsi, on peut s'attendre à ce que la condition sur le type de media soit complètement dépréciée dans le futur. A l'heure actuelle, nous pouvons choisir parmi les types de media suivants :

- all : Valeur par défaut. Nos règles vont s'appliquer à tous les appareils ;
- screen : Nos règles ne vont s'appliquer qu'aux appareils dotés d'un écran ;
- printer : Nos règles ne s'appliqueront que pour les imprimantes ;
- speech : Nos règles ne s'appliqueront qu'aux liseurs d'écran qui sont capable de rendre le contenu d'une page de manière sonore.

Nous allons également pouvoir inverser la valeur logique d'un test avec le mot clef not. En plaçant le mot clef not avant le type de media, nos règles s'appliqueront à tous les appareils media sauf celui spécifié.

Ensuite, nous allons pouvoir créer des conditions sur les caractéristiques du media. Notez déjà qu'on peut créer autant de conditions sur des caractéristiques différentes que l'on souhaite.

Nous allons devoir entourer chaque condition sur une caractéristique media avec un couple de parenthèses et allons pouvoir séparer deux conditions avec les mots clefs and (et) ou or (ou).

Dans le cas où on utilise and, les deux conditions devront être vérifiées. Dans le cas où on utilise or, il suffira qu'une condition soit vérifiée.

Il existe de nombreuses caractéristiques sur lesquelles on peut effectuer nos tests. Cependant, en pratique, nous utiliserons généralement des conditions de taille pour distinguer entre différents appareils et utiliserons pour cela les propriétés width, min-width et max-width.

Sachez toutefois qu'on peut également conditionner l'application de nos styles CSS à la hauteur d'un media, sa résolution, son processus de scan, à la présence d'un appareil de pointage parmi les mécanismes de saisie et sa précision ou encore à la capacité de l'appareil à survoler les éléments.

Utilisation pratique des Media Queries

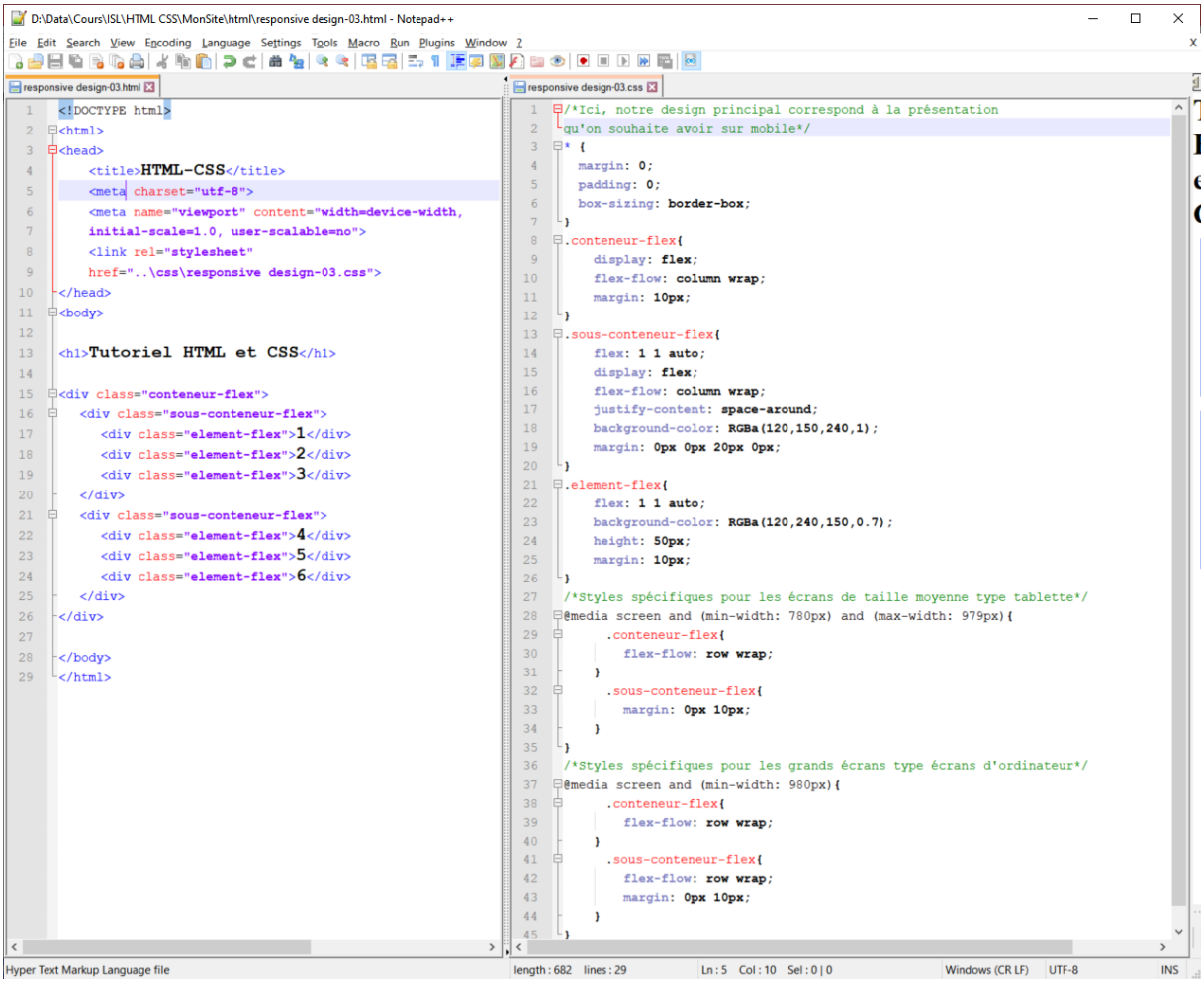
Depuis quelques années, la majorité des recherches web se font sur mobile. C'est la raison principale qui a amené Google à aujourd'hui indexer la version mobile des sites Internet et non plus leur version bureau. Cela fait qu'il est indispensable d'avoir une version mobile performante aujourd'hui.

Pour cette raison, il est considéré comme une bonne pratique aujourd'hui de créer son site en version mobile d'abord puis d'utiliser les Media Queries pour modifier la disposition du code pour les écrans d'ordinateurs ou de tablettes.

Fonctionner de cette manière peut vous sembler tout à fait logique si vous n'avez jamais développé auparavant mais je peux vous assurer que c'est une façon de procéder qu'il est difficile de faire admettre comme norme pour des développeurs expérimentés et habitués à créer des sites version bureau puis à les décliner pour mobile.

Dorénavant, nous travaillerons comme cela : on construira la version mobile de nos pages d'abord et nous utiliserons les Media Queries pour adapter nos pages pour des grands écrans.

Illustrons cela immédiatement avec un exemple :



responsive design-03.html

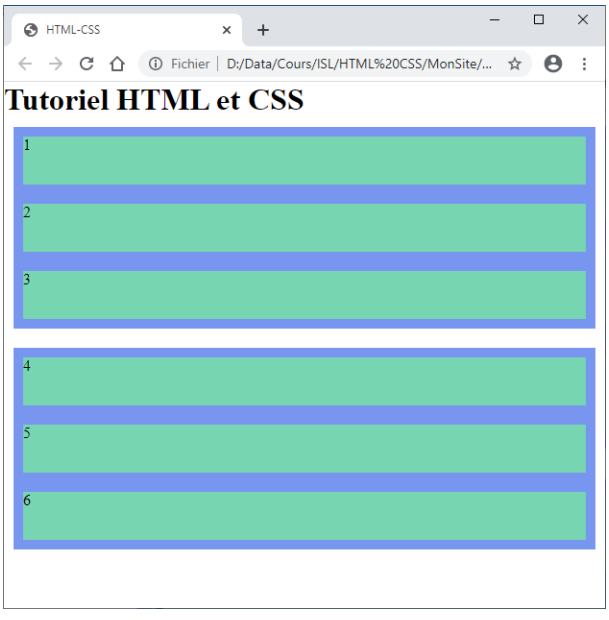
```
<!DOCTYPE html>
<html>
<head>
    <title>HTML-CSS</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1.0, user-scalable=no">
    <link rel="stylesheet"
    href="..\css\responsive design-03.css">
</head>
<body>

<h1>Tutoriel HTML et CSS</h1>

<div class="conteneur-flex">
   <div class="sous-conteneur-flex">
      <div class="element-flex">1</div>
      <div class="element-flex">2</div>
      <div class="element-flex">3</div>
   </div>
   <div class="sous-conteneur-flex">
      <div class="element-flex">4</div>
      <div class="element-flex">5</div>
      <div class="element-flex">6</div>
   </div>
</div>

</body>
</html>
```

responsive design-03.css

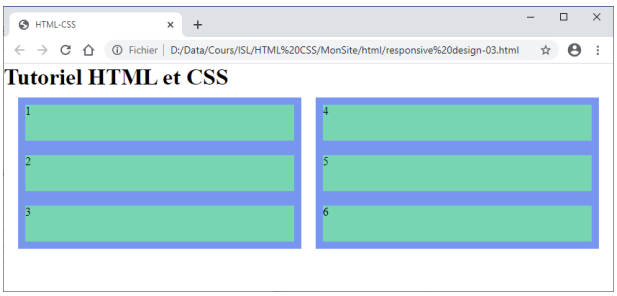
```
/*Ici, notre design principal correspond à la présentation
qu'on souhaite avoir sur mobile*/
* {
   margin: 0;
   padding: 0;
   box-sizing: border-box;
}
.conteneur-flex{
    display: flex;
    flex-flow: column wrap;
    margin: 10px;
}
.sous-conteneur-flex{
    flex: 1 1 auto;
    display: flex;
    flex-flow: column wrap;
    justify-content: space-around;
    background-color: RGBa(120,150,240,1);
    margin: 0px 0px 20px 0px;
}
.element-flex{
    flex: 1 1 auto;
    background-color: RGBa(120,240,150,0.7);
    height: 50px;
    margin: 10px;
}
/*Styles spécifiques pour les écrans de taille moyenne type tablette*/
@media screen and (min-width: 780px) and (max-width: 979px){
     .conteneur-flex{
       flex-flow: row wrap;
    }
     .sous-conteneur-flex{
       margin: 0px 10px;
    }
}
/*Styles spécifiques pour les grands écrans type écrans d'ordinateur*/
@media screen and (min-width: 980px){
     .conteneur-flex{
       flex-flow: row wrap;
    }
     .sous-conteneur-flex{
       flex-flow: row wrap;
       margin: 0px 10px;
    }
}
```

Hyper Text Markup Language file — length : 682 lines : 29 — Ln : 5 Col : 10 Sel : 0 | 0 — Windows (CR LF) — UTF-8 — INS

Résultat version mobile :



Résultat version tablette :



Résultat version bureau :

Dans cet exemple, nous travaillons avec trois niveaux de div : un premier élément conteneur global, deux sous-conteneurs et des div qui servent d'éléments dans ces sous conteneurs.

En CSS, on commence donc par créer nos styles mobiles. On va ici appliquer un display : flex à notre conteneur et à nos sous conteneurs afin d'en faire des conteneurs flexibles.

Nos sous conteneurs vont donc à la fois être des conteneurs flexibles et des éléments flexibles. Comme les écrans de mobile sont les plus petits, on choisit une organisation des éléments verticale en appliquant un flex-flow : column wrap à la fois à notre conteneur flexible principal et à nos sous conteneurs.

On applique également des couleurs de fond et des marges externes à tous les div afin de bien pouvoir les différencier visuellement.

Ensuite, on définit deux règles @media qui vont s'adresser à différents appareils.

Notre première règle @media est la suivante : @media screen and (min-width: 780px) and (max-width: 979px){}. Cette règle va cibler les appareils media dotés d'un écran dont la largeur est comprise entre 780px et 979px.

Si un utilisateur utilise un appareil qui remplit ces conditions, alors les styles CSS indiqués dans la règle @media seront appliqués. Ici, notre règle s'adresse aux écrans de taille moyenne de type tablette.

On va vouloir pour ces écrans disposer nos sous conteneurs en ligne mais conserver un affichage des éléments dans les sous conteneurs en colonne. On va donc seulement modifier la direction du conteneur principal.

Ici, il faut bien comprendre que les styles de notre règle @media vont être traités de manière prioritaire et surcharger les styles définis globalement si les conditions relatives à l'appareil sont remplies.

Cependant, les autres propriétés définies globalement et qui ne sont pas reprécisées dans notre règle @media vont continuer à s'appliquer normalement (comme la couleur de fond par exemple ou la direction des sous conteneurs et etc).

Finalement, notre deuxième règle @media s'adresse aux grands écrans. On va ici vouloir afficher tous nos éléments en ligne. Nous allons donc attribuer un flex-flow : row wrap à la fois à notre conteneur flexible principal et à nos sous conteneurs flexibles.

Images responsive

Grâce aux Media Queries, nous avons pu réorganiser nos éléments HTML en créant des règles CSS spécifiques qui se s'appliquaient qu'à certaines tailles d'écran.

Cependant, on ne va pas pouvoir « réorganiser » une image en CSS. Nous allons bien évidemment pouvoir placer notre image dans un conteneur en HTML puis utiliser par exemple le flexbox pour rendre l'image « responsive » mais cela ne va pas être une solution optimale et ceci pour deux raisons :

- En n'utilisant qu'une seule image, nous allons être obligés de choisir la version la plus grande possible de celle-ci afin qu'elle ne pixellise pas lorsqu'un utilisateur l'affiche sur un grand écran. Or, plus une image est grande et plus elle est lourde : nous allons donc imposer aux mobiles le téléchargement d'une image très lourde sans raison ce qui va ralentir la vitesse d'affichage de nos pages ;
- En n'utilisant qu'une seule image, il est fort probable que l'image ne rende pas bien sur mobile car comme celle-ci sera plus petite à cause de la taille de l'écran le sujet dans l'image va apparaitre lointain ou comme dézoomé.

Ces deux problématiques font qu'on aimerait pouvoir proposer différentes versions d'une image pour différents types d'appareil. Nous allons voir comment faire dans ce chapitre.

Pixels physiques, pixels CSS et écrans retina

Nous l'avons vu dans le chapitre sur le viewport : un pixel n'est pas toujours égal à un pixel ! Comprendre cela va être très important pour afficher des images qui vont s'afficher correctement sur tous les appareils.

Ici, retenez que les écrans retina ont généralement une résolution deux fois plus importante que les écrans standards. Cela signifie que chaque « pixel retina » est l'équivalent de 4 « pixels standards » (2 en largeur, 2 en hauteur).

Ainsi, pour qu'une image s'affiche correctement sur un écran retina, il faudra qu'elle soit deux fois plus grande que l'écran sur lequel elle doit s'afficher (1960x 980px par exemple pour un affichage sur un écran retina de 980x 490px).

Notez également ici que certains écrans retina possèdent une résolution 3 fois plus importante qu'un écran standard, auquel cas il faudra une image 3 fois plus grande et etc.

Une première solution : utiliser des images SVG

La première solution, qui semble la plus évidente pour gérer les différentes tailles d'écran et les différentes résolutions est d'utiliser des images au format SVG pour Scalable Vector Graphic.

Comme le nom l'indique, ces images sont vectorielles, ce qui signifie qu'on va pouvoir les agrandir ou les rapetisser à l'infini sans perte de qualité.

Cependant, cela ne résout qu'une partie du problème puisqu'en n'utilisant qu'une seule image pour toutes les versions de notre site, cette image risque d'apparaitre comme trop imposante pour la version bureau ou trop dézoomée pour la version mobile.

De plus, souvent, vous serez obligés d'intégrer des photos ou images d'un format différent comme jpeg ou png.

Proposer plusieurs versions d'une image en utilisant les attributs HTML srcset et sizes

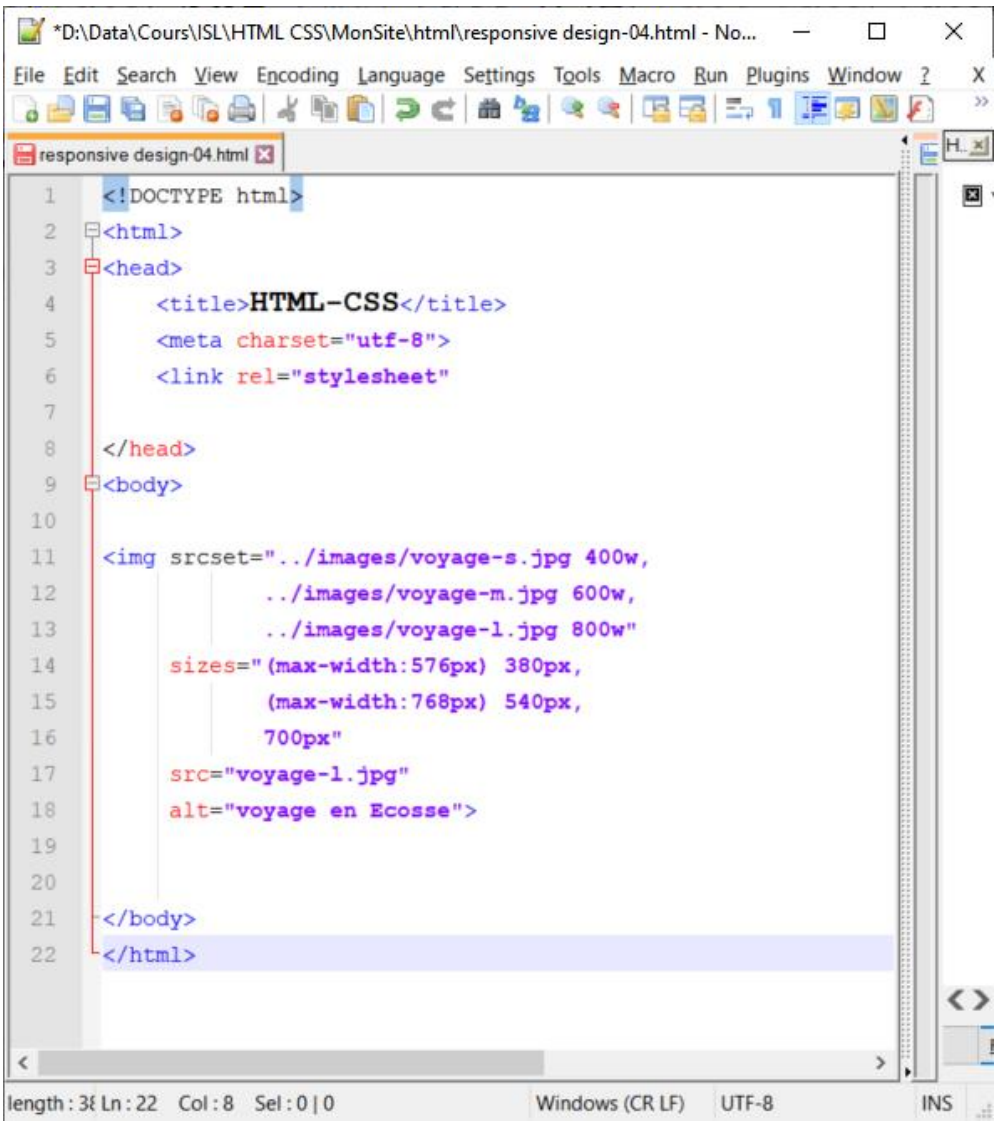
Nous allons pouvoir ajouter un attribut srcset dans notre élément img qui va nous permettre de fournir plusieurs sources d'images (c'est-à-dire concrètement de fournir plusieurs images différentes) au navigateur parmi lesquelles choisir.

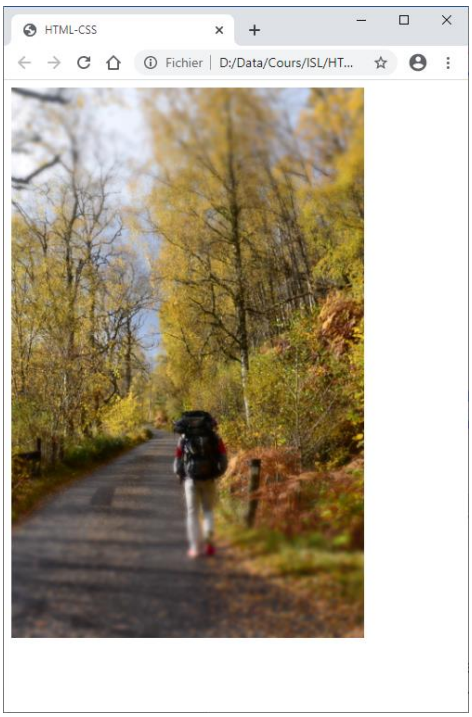
Ce qui va nous intéresser ici va être de proposer plusieurs versions d'une même image avec des tailles et des cadrages différents. Pour cet exercice, je vais utiliser trois versions d'une même image que j'ai prédécoupé avant :

- Une version complète « L » de l'image de dimensions 800x 625px que l'on va servir aux grands écrans ;
- Une version « M » rognée avec un premier centrage sur le sujet pour les écrans de taille moyenne ;
- Une version « S » de dimensions 400x 625px complètement centrée sur le sujet pour les petits écrans.

Voici à quoi ressemblent les trois photos :

L'attribut srcset va être accompagné d'un attribut sizes qui va nous permettre de préciser un ensemble de conditions relatives au terminal utilisé par un utilisateur (généralement des conditions de tailles de fenêtre) et d'indiquer la largeur que doit occuper l'image dans chaque situation.


*D:\Data\Cours\ISL\HTML CSS\MonSite\html\responsive design-04.html - No...
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
responsive design-04.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>HTML-CSS</title>
5 <meta charset="utf-8">
6 <link rel="stylesheet"
7
8 </head>
9 <body>
10
11 <img srcset="../images/voyage-s.jpg 400w,
12 ../images/voyage-m.jpg 600w,
13 ../images/voyage-l.jpg 800w"
14 sizes="(max-width:576px) 380px,
15 (max-width:768px) 540px,
16 700px"
17 src="voyage-l.jpg"
18 alt="voyage en Ecosse">
19
20
21 </body>
22 </html>
length : 3 Ln : 22 Col : 8 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS



Ici, nous renseignons chacune des versions de notre image dans l'attribut srcset avec leur valeur intrinsèque en unités w qui sont un équivalent pixel. Pour faire très simple, nous indiquons la largeur de chacune de nos images en pixels avec l'unité w.

Ensuite nous allons préciser dans l'attribut size un ensemble de conditions et la largeur de l'espace que doit occuper l'image dans l'écran si une condition est vérifiée.

Les conditions seront généralement des conditions sur la taille de la fenêtre et vont être le strict équivalent de nos Media Queries CSS. Par exemple la condition (max-width : 576px) va être vérifiée pour toutes les tailles de fenêtre inférieures à 576px et l'image devra alors être affichée dans un espace de 380px de large.

Si cette condition est vérifiée, alors le navigateur utilisera l'image dont les dimensions sont le plus proche de l'espace dans lequel elle devra être affichée.

Notez que l'ordre d'écriture des conditions dans sizes compte puisque c'est toujours la première condition vérifiée qui va être retenue. Ainsi, nous déclarerons toujours nos conditions de la plus restrictive à la moins restrictive afin que le navigateur récupère la meilleure version à chaque fois.

Le navigateur va donc ici commencer par calculer la taille de l'écran de vos visiteurs puis passer en revue les conditions données jusqu'à ce qu'une condition soit vérifiée. Dès que c'est le cas, le navigateur va noter la largeur de la place que doit occuper l'image et va charger l'image dans srcset dont la taille est la plus proche de la largeur précisée dans notre condition.

Attention cependant ici : l'attribut srcset ne donne qu'une indication de préférence au navigateur et celui-ci est libre de l'ignorer pour charger la version de l'image qu'il souhaite. Ainsi, il est possible que

le navigateur d'un visiteur ne charge pas la version souhaitée dans le cas où, par exemple, le navigateur possèderait déjà une version avec une meilleure résolution de l'image en cache.

En effet, la plupart des navigateurs comprennent bien que l'attribut srcset est utilisé à des fins d'optimisation. Or, si le navigateur possède déjà une version de qualité supérieure cachée d'un média, c'est-à-dire une version déjà téléchargée et prête à l'affichage, alors il est normal qu'il l'affiche puisque cela optimisera le rendu (supposément) et les performances.

Faites donc bien attention de votre côté si vous testez srcset à le tester en navigation privée ou à supprimer le cache de votre navigateur afin de bien voir l'image changer en fonction de la taille de l'écran.