UNIVERSITE PARIS 8
Vincennes - Saint-Denis

XHTML, HTML5, DOM et CSS

M2 G2M, Univ. Paris 8
par Isis TRUCK

Inspiré de diverses sources
dont http://xhtml.le-developpeur-web.com et
http://mymi.domy.be/LTP/index.html

Historique

- *XHTML (eXtensible HyperText Markup Language)*
  - est issu du HTML (en est une évolution) et se fonde sur la syntaxe XML
  - Langage de balises pour traiter de la **forme** (cf. fond et forme)
- HTML
  - créé en 1998 par le W3C (World Wide Web Consortium, organisme de normalisation) pour créer facilement des pages web
  - les balises sont **figées** (impossible d'en créer de nouvelles, selon ses besoins)
- XML (*eXtensible Markup Language)* :
  - langage de balises pour écrire des informations de manière "universelle »
  - est une version simplifiée de SGML (Standard Generalized Markup Language, datant de 1986, qui était un des 1ers langage de description à balises)
  - les balises sont **créées par l'utilisateur**
- On distingue deux types de langages :
  - les **langages interprétés** (l'interpréteur lit le code (les instructions) au fur et à mesure)
  - les **langages compilés** (le compilateur lit la totalité du code et transforme le programme lu en un nouveau programme, dit *exécutable*)
- HTML, XHTML et XML, tous héritiers de SGML, sont des langages interprétés

Exemples de balises XHTML

- `<a>`
  - Signifie « anchor » (ancre)
  - Utilisation: `<a attribut1="…" attribut2="…" …>TEXTE</a>`

  Exemple d'attribut : `href` (« hypertext reference »)

  ```
  <a href="http://www.univ-paris8.fr">Mon université</a>
  ```
- `<h1>`
  - Signifie « heading 1 » (titre de 1er niveau)
  - Utilisation: `<h1>Mon titre entête</h1>`
- `<br />`
  - Signifie « break » (pour *casser* une ligne = retour chariot)
  - Utilisation: `<p>Ma phrase<br />sur 2 lignes</p>`
  - Il s'agit d'une balise *autofermante* : ne pas oublier le *space* – *slash* avant de fermer le chevron (>)

Les 5 règles de syntaxe XHTML

- Toute balise ouvrante doit obligatoirement être **fermée**
- Les balises et les attributs doivent être écrits en **minuscules**
- Les valeurs des attributs doivent être entre **guillemets** ou ***quotes*** **(apostrophes)**
- Chaque attribut doit avoir une **valeur**
- Chaque élément doit être **imbriqué correctement**

Arbre logique d'un document

- représentation permettant de comprendre la structure d'un document *(prendre l'exemple de la diapo 14)*
- permet de bien comprendre comment ***sélectionner*** tel ou tel élément dans le document, que ce soit en CSS ou en Javascript.
- représentation très intuitive issue du vocabulaire généalogique :
  - **ROOT** : Racine de l'arbre. C'est l'élément de base du document XML; pour un document XHTML, c'est `html`; c'est l'élément qui apparaît dans l'instruction DOCTYPE.
  - **NODE** : Un nœud est un élément de l'arbre, une balise.
  - **PARENT** : Un élément P est dit parent de E s'il est lié par une ou plusieurs branches à cet enfant et s'il est plus haut d'un niveau dans la hiérarchie (***parent direct***) ou de plusieurs niveaux (***parent indirect*** ou grand-parent...); un nœud n'a qu'un seul parent direct. Par exemple, `html` est le parent de `body`.

Arbre logique (suite)

- **CHILD, CHILDREN** : Un élément E est dit enfant de P s'il est ***directement*** lié par une branche à son parent, et plus bas d'un niveau dans la hiérarchie; par exemple, `body` est un enfant de `html`. On parle aussi de petits-enfants, d'enfants directs ou indirects.
- **Descendant** : Les enfants, petits-enfants, etc. donc tous les éléments qui sont liés à un parent donné par une succession de branches de l'arbre sont les descendants de ce parent. Dans un document, tous les nœuds de l'arbre sont descendants de la racine de l'arbre.
- **LEAF : Feuille**. C'est un nœud de l'arbre qui n'a pas d'enfant. En XML/XHTML, ce sera généralement du texte (noté #text), ou bien un élément vide ou simple : c'est la même chose, ce sont ces balises qui n'ont pas de marqueur de fermeture.

- La visualisation de l'arbre peut se faire sur papier avec la racine en haut et à gauche, ou à l'aide d'outils comme l'inspecteur DOM (cf. plus loin)

XHTML et sémantique

- Sémantique = sens donné à un élément de la page web
- Ici, **aucun sens** donné :
  - `<div>Chapitre 1 : Le xhtml et la sémantique</div>`
    `<div>Le xhtml doit donner du sens !</div>`
- Là, il y a **un sens** :
  - `<h2>Chapitre 1 : Le <abbr>xhtml</abbr> et la sémantique</h2>`
    `<p>Le <abbr>xhtml</abbr> doit donner du sens !</p>`
- On donne du sens pour mieux séparer le fond de la forme :
  - pour **améliorer l'accessibilité** : ex. un appareil de synthèse vocal saura ainsi que tel élément est le titre de la page, celui-là le sous-titre n°1 et ce texte-ci le premier paragraphe du sous-titre n°1. Il saura aussi que ces 5 lettres sont une abréviation, etc.
  - pour **améliorer son référencement** : un moteur de recherche ne peut savoir ce que tel ou tel élément de la page représente, sans aide. Entourer un élément par une balise spécifique permet d'indiquer au *robot* ce que représente cet élément. Le moteur pourra ainsi mieux "comprendre" le sens d'une page et mieux la référencer.

XHTML et accessibilité

- XHTML permet d'améliorer l'accessibilité des sites
- Accessibilité = capacité à accéder un site Internet via **n'importe quel** logiciel capable de naviguer sur un site Web:
  - les navigateurs web traditionnels (Firefox, Internet Explorer, Safari...) et les navigateurs des téléphones mobiles (Android Browser, Opera Mini, Safari Mobile, Chrome, Google Goggles…)
  - les appareils de synthèse vocale utilisées par les handicapés,
  - le navigateur d'un réfrigérateur (domotique),
  - le navigateur intégré au tableau de bord d'une voiture (informatique embarquée)...
- Utilité du XHTML : réaliser **un et un seul** site, de manière à **ne pas décliner** autant de sites qu'il existe de logiciels capables de naviguer sur des pages Web => *responsive Web design* (cf.slide 58)

XHTML ≠ CSS

- Ne pas confondre XHTML et CSS
  - les CSS sont des styles qui définissent la **présentation** de la page
  - alors que XHTML définit la **structure** de la page
- Structure et présentation sont **différents** mais **complémentaires**
- Dans le code suivant `<abbr>XHTML</abbr>`,
  - la balise `<abbr>` définit la structure de la page et sa sémantique en spécifiant que XHTML est une abréviation
  - grâce aux styles CSS, on peut appliquer une présentation particulière en affichant systématiquement en gras, par exemple, tous les mots entourés des balises <abbr> et </abbr> (voir la partie du cours dédiée aux CSS)

Doctypes XHTML

- Le `doctype` est la DTD (*Document Type Declaration :* déclaration de type de document) de la page web
- Il indique quelles sont **les règles à suivre**.
- Il est déclaré dans la 1re ligne d'instruction d'un fichier XHTML 1.0
- 3 `doctype` sont possibles :
  - XHTML 1.0 Transitional
  - XHTML 1.0 Frameset
  - **XHTML 1.0 Strict** (préféré)

XHTML 1.0 Transitional

- Instruction : `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
- respecte les 5 principes de base du XHTML
- accepte les balises et les attributs agissant sur la présentation de la page web. Exemple : <font>, , …
- accepte d'utiliser l'attribut "target" sur les balises <a>
- accepte d'utiliser la balise <iframe>
- mais ces balises et attributs sont à déconseiller d'un point de vue conceptuel. Il faut toujours bien séparer la structure de la page (XHTML) de sa présentation (CSS)
- => Transitional : déconseillé

XHTML 1.0 Frameset

- Instruction : `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">`
- respecte les 5 principes de base du XHTML
- accepte les balises et les attributs agissant sur la présentation de la page web. Exemple : <font>, , …
- accepte d'utiliser l'attribut "target" sur les balises <a>
- accepte d'utiliser la balise <iframe>
- mais ces balises et attributs sont à déconseiller d'un point de vue conceptuel.
- balise <body> est remplacée par la balise <frameset> qui contient les balises <frame>
- => Frameset : déconseillé

XHTML 1.0 Strict

- Instruction : `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
- respecte les 5 principes de base du XHTML
- n'accepte pas les balises et les attributs agissant sur la présentation de la page web. (<font>, , …)
- n'accepte pas d'utiliser l'attribut "target" sur les balises <a>
- n'accepte pas d'utiliser la balise <iframe>
- => Strict : préféré

Template pour une page en XHTML 1.0 Strict

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
              "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
     <title>Titre de la page</title>
     <meta name="keywords" lang="fr" content="kw1,kw2" />
     <meta name="description" content="Description de la page" />
     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
     <meta http-equiv="Content-Language" content="fr" />
     <meta http-equiv="Content-Script-Type" content="text/javascript" />
     <link href="style.css" rel="stylesheet" type="text/css" />
</head>
<body>

</body>
</html>
```

Template pour une page en XHTML 1.1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
   <head>
        <title>Document XHTML minimal</title>
   </head>
   <body>
   ...
   </body>
</html>
```

- NB: le choix d'***encodage*** est ici le jeu de caractères **UTF-8**, l'encodage recommandé
- ***=> Ce choix écrit dans le code implique que le fichier soit effectivement encodé en UTF-8*** *(sans BOM, si choix possible)* ***au moment de l'enregistrement !***

XHTML 1.0 et XHTML 1.1

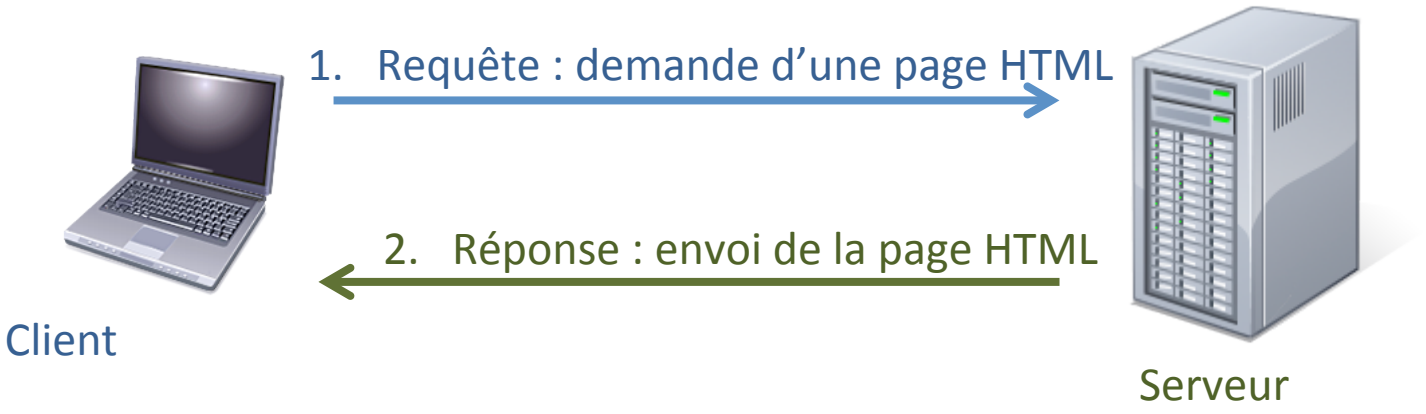
- On parle de 2 versions : XHTML 1.0 et XHTML 1.1
- Pour le moment, **on code en XHTML 1.0** (site du W3C est en XHTML 1.0) : il faut bien préciser le bon `doctype` dans le fichier
- Différences en 1.0 et 1.1 :
  - XHTML 1.0 accepte d'être interprété comme un document HTML.
    C'est pour cette raison que son *type mime* est "`text/html`".
    ```
    <meta http-equiv="Content-Type" content="text/html;
    charset=iso-8859-1" />
    ```
  - XHTML 1.1 **n'accepte pas** d'être interprété comme un document HTML.
    Pour cette raison, son type mime **doit être** "**`application/xhtml + xml`**".
    ```
    <meta http-equiv="Content-type" content="application/xhtml + xml ;
    charset=ISO-8859-1" />
    ```
- La plupart des serveurs sont configurés pour envoyer du contenu de type "text/html" lorsqu'il s'agit de fichiers ".php", ".html", ".asp"... Il faut donc **modifier les données envoyées par le serveur**, soit directement sur le serveur, soit *via* un **langage serveur** (voir diapo suivante).
  Exemple en langage php : `header('Content-type:application/xhtml+xml');`
- NB: le type mime "application/xhtml + xml" n'est pas supporté par tous les navigateurs.

Langage serveur ?

- encore appelé **langage de script côté serveur** (*server-side scripting*)
- langage de programmation qui interagit avec un serveur HTTP pour produire une page Web dynamique
- est **interprété par le serveur** qui génère ensuite du code (XHTML, CSS, Javascript…) interprété **sur l'ordinateur du visiteur** (= client)
- nécessaire pour utiliser une base de données
- indispensable pour la majorité des scripts complexes
- Exemples de langage serveur:
  - PHP (« *Personal Home Page* » puis « *PHP: Hypertext Preprocessor* »)
  - ASP (*Active Server Pages*, Microsoft)
  - CGI (*Common Gateway Interface*)
  - JSP (*Java Server Pages*)

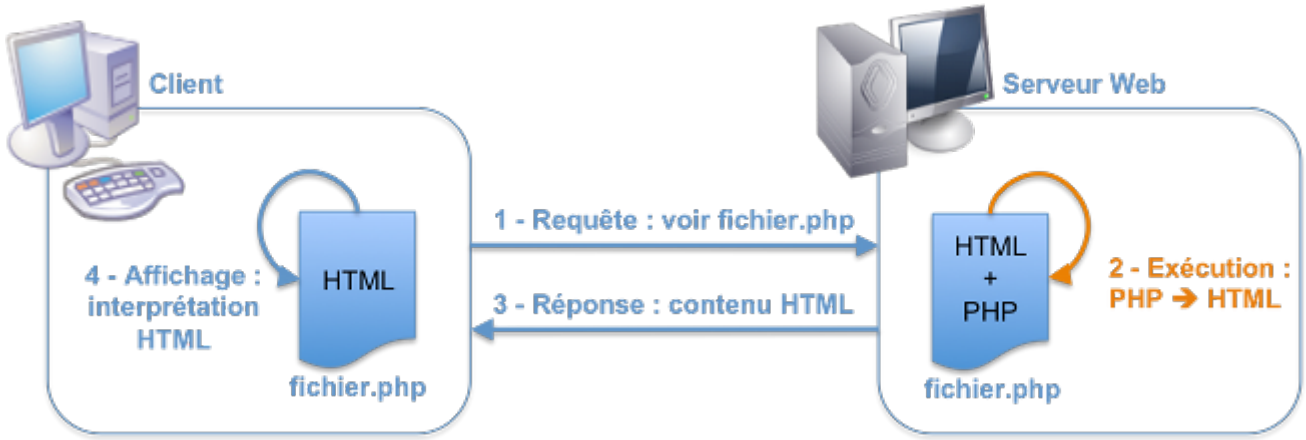
Serveur, client, Kesako?

- Serveur : dispositif (hardware ou software) qui fonctionne en permanence et qui sait répondre à des requêtes de clients (selon un certain **protocole réseau**)
- Requêtes peuvent être pour
  - consulter un courrier électronique (serveur de mail)
  - accéder à un contenu du WWW (serveur HTTP)
  - accéder à un fichier, à une base de données, etc.
- Client: logiciel qui envoie les requêtes. Soit le logiciel est manipulé par un humain, soit il est autonome (c'est un ***bot*** – vient de *robot*)



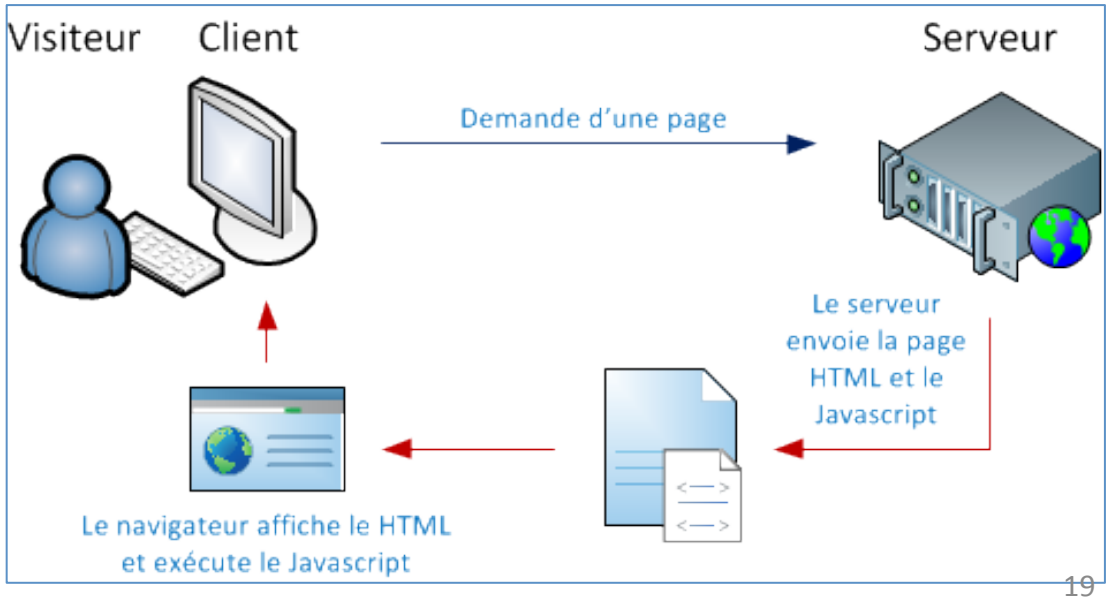
Dialogue client/serveur: autres cas

- Cas PHP



Source: www.enseignement.polytechnique.fr

- Cas Javascript



Source: www.siteduzero.com

Exemple : affichage de la date

	Langage serveur (PHP)	Langage client (JavaScript)

Page créée au départ	`Nous sommes le <?php echo date('j/m/Y'); ?>`	`<script type="text/javascript">`
		`today = new Date();`
		`document.write("Nous sommes le ", today.getDate(), "/", today.getMonth()+1, "/", today.getFullYear());`
		`</script>`

Code source de la page envoyée au navigateur	Nous sommes le 9/10/2018	`<script type="text/javascript">`
		`today = new Date();`
		`document.write("Nous sommes le ", today.getDate() ,"/", today.getMonth()+1, "/", today.getFullYear());`
		`</script>`

Résultat dans le navigateur	Nous sommes le 9/10/2018	Nous sommes le 9/10/2018

=> les deux langages donnent exactement le même résultat, si toutefois le client supporte et a activé le JavaScript.

Les 3 types de balises en XHTML

- En XHTML, il existe **3 types de balises** :
  - les balises de type **BLOC**
  - les balises de type **EN LIGNE**
  - les balises de type **AUTOFERMANTE**

Balises de type **BLOC**

- boites d'éléments qui peuvent contenir des éléments de type feuille (comme du texte noté `#text` ou des commentaires notés `#comment`), des balises de type **EN LIGNE**, des balises de type **AUTOFERMANTE** et d'autres balises de type **BLOC**.
- Les balises de type BLOC se positionnent toujours **les unes en dessous des autres**.
  Exemple : `<div>J'aime Paris 8</div><div>Vive Paris 8</div>`
  affichera :

  J'aime Paris 8

  Vive Paris 8
- Leur **largeur native** (= par défaut) est **celle de l'élément qui les contient**.
  Exemple 1 : La balise occupera toute la largeur de la page (car parent = body)

```
<body>
     <div>Vive Paris 8</div>
</body>
```

- Exemple 2 : La balise enfant occupera toute la largeur de la balise parent.

```
<div>
     <div>Vive Paris 8</div>
</div>
```

Balises de type **BLOC** (suite)

- Liste des balises BLOC:

<address> , <blockquote> , <body> , <dd> , , <dl> , <dt> , <fieldset> , <form> , <h1> , <h2> , <h3> , <h4> , <h5> , <h6> , <head> , <hr /> , <html> , <li> , <map> , <noscript> , <ol> , <p> , <pre> , <script> , <style> , <table> , <tbody> , <td> , <tfoot> , <th> , <thead> , <tr> ,

Balises de type **EN LIGNE**

- ne peuvent contenir que
  - des éléments *feuilles*,
  - d'autres balises EN LIGNE,
  - mais **pas de balises de type BLOC**.
- Les textes contenus dans des balises EN LIGNE se positionnent toujours les uns à côté des autres (contrairement aux BLOC)
- Les balises de type EN LIGNE n'ont pas de dimension (aucune largeur ni hauteur)
- Liste des balises EN LIGNE:

```
<a> , <abbr> , <acronym> , <area /> , <bdo> , <br />,
<button> , <caption> , <cite> , <code> , <del> ,
<dfn> , <em> , <img /> , <input /> , <ins> , <kbd> ,
<label> , <legend> , <link /> , <meta /> , <object> ,
<optgroup> , <option> , <param /> , <q> , <samp> ,
<select> , <span> , <strong> , <sub> , <sup> ,
<textarea> , <title> , <var>
```

Balises de **type AUTOFERMANTE**

- balises qui sont ouvrantes et fermantes à la fois
- elles n'ont donc **pas de contenu**
- on **ferme** la balise en ajoutant un espace et un slash (/) à la fin de la balise
- les balises AUTOFERMANTES sont :
  - soit des balises de type **BLOC** (exemple : <hr />),
  - soit des balises de type **EN LIGNE** (exemple : <img />).
- Liste des balises de type AUTOFERMANTE

  <area /> ,
 , <hr /> , <img /> , <input /> ,
  <link /> , <meta /> , <param />

Conseils pour bien structurer le document XHTML

- ne pas utiliser les balises et attributs obsolètes
  - Balises dépréciées : <basefont> , , <font> , <s> , <strike> , <u>
  - Attributs dépréciés (par balise) :

    <a> (les attributs **target**)
    <body> (les attributs **alink**, **background**, **bgcolor**, **link**, **vlink**, **text**)
    <caption> (les attributs **bgcolor**, **height**, **nowrap**, **width**)
    <hr /> (les attributs **clear**, **noshade**, **size**, **width**)
    <img /> (les attributs **border**, **hspace**, **vspace**)
     (les attributs **compact**, **type**)
    <object> (les attributs **border**, **hspace**, **vspace**)
     (les attributs **star**, **value**)
    <pre> (les attributs **width**)
    <table> (les attributs **bgcolor**, **height**, **nowrap**, **width**)
    <tbody> (les attributs **bgcolor**, **height**, **nowrap**, **width**)
    <td> (les attributs **bgcolor**, **height**, **nowrap**, **width**)
    <tfoot> (les attributs **bgcolor**, **height**, **nowrap**, **width**)
    <thead> (les attributs **bgcolor**, **height**, **nowrap**, **width**)
    <th> (les attributs **bgcolor**, **height**, **nowrap**, **width**)
    <tr> (les attributs **bgcolor**, **height**, **nowrap**, **width**)

Conseils (suite)

Utiliser les **commentaires**

- Syntaxe d'un commentaire : balise spéciale commençant par <! -- et se terminant par -->
- Pourquoi commenter ?
  - Pour expliquer et clarifier la structure du document notamment lors d'imbrications complexes de balises et lorsque le nombre de lignes de code est important.
  - Pour permettre la transmission du code (on écrit **pour être relu** !)
- les commentaires ne sont pas interprétés par les navigateurs
- Exemple :

```
<!-- Debut de la zone de recherche -->
     <div>
          …
     </div>
<!-- Fin de la zone de recherche -->
```

Conseils (suite)

- Utiliser correctement les attributs '**id**' et '**class**'
  - les balises XHTML peuvent être nommées via 2 attributs : **id** et/ou **class**
  - => on peut ensuite appliquer des styles CSS aux balises ainsi nommées
- L'attribut **id** permet d'identifier une balise d'un document XHTML **de manière unique**
- Deux balises ne doivent pas avoir 2 attributs **id** ayant la même valeur au sein du même document.
  Exemple :

```
<div>
   <div id='peinture'>
      <p>Vermeer, en peignant La jeune fille à la perle…</p>
   </div>
</div>
```

  Dans un fichier CSS externe, on pourra ainsi modifier la couleur de font du via la syntaxe suivante :

```
#peinture {background-color:#ff0000;}
```

Conseils (suite)

- L'attribut **class** permet de **nommer un groupe** de balises au sein d'un même document XHTML.
- => Plusieurs balises peuvent ainsi avoir un attribut **class** ayant la **même valeur**.

  Exemple :

```
<div class='peinture'>
  <div class='peinture'>
    <p>Vermeer, en peignant La Jeune fille à la perle…</p>
  </div>
</div>
```

  Dans un fichier CSS, on pourra modifier la couleur de fond des 2 :

```
.peinture {background-color:#ff0000;}
```

Conseils (fin)

- valider son code avec des validateurs
  - Un validateur XHTML est une application qui permet de valider les pages web au format souhaité : XHTML strict 1.0, XHTML transitional 1.0, etc.
  - Exemples de validateurs (en ligne) :
    - http://validator.w3.org/
    - http://www.validome.org/lang/fr
  - Les IDE (Integrated Development Environment) comprennent généralement un validateur:
    - Netbeans
    - Dreamweaver (payant…)

DOM (Document Object Model)

- L'outil ***DOM Inspector*** de Firefox, qui se trouve dans le menu Tools/Outils du navigateur, génère la visualisation relative à la page courante.
- Le DOM Inspector offre plusieurs visualisations relatives au document, par exemple celle liée au DOM Nodes : l'arbre logique à gauche (choisir ***Document - DOM Nodes***, dans la liste en haut de ce volet), les informations sur les nœuds de l'arbre (choisir ***Object - DOM Node***, dans la liste en haut de ce volet).
- Ces informations renseignent sur le type de nœud, les attributs associés et leur valeur

Vocabulaire sur le document

- Les 5 règles de syntaxe résument tout ce qui doit être connu pour écrire un document ***bien formé***.
- Un document est ***valide*** quand les règles d'imbrication de balises et d'utilisation des attributs sont appliquées
- Un document est ***correct*** quand la sémantique est respectée
- Un document est ***lisible*** si l'on *indente* le code correctement et que l'on commente.

Insertion d’images

- L'image introduite par le code XHTML doit avoir un rôle **sémantique** : illustration d'un texte, galerie d'images, image pour la navigation, image réactive, un bouton image...
- Dès que l'image est un élément décoratif, elle doit être introduite au moyen de CSS, en tant qu'image de fond pour un bloc ou une barre horizontale, une variante image de puce de liste, une décoration de bouton de formulaire,...
- Une image doit être légère (bande passante) : résolution doit être juste suffisante
- Adobe Photoshop offre la possibilité d'enregistrer pour le Web. La résolution utile pour un affichage à l'écran est de 72 dpi (elle doit être supérieure pour l'impression).
- La taille des images les plus grandes doit rester de l'ordre de la centaine de ko; elle doit être nettement plus faible pour les petites images ! On veillera à ne pas créer de pages avec un poids total d'images excessif. Le mécanisme des miniatures doit en tout cas être utilisé pour ne pas forcer l'utilisateur à télécharger des images de taille importante.

Insertion d’images (suite)

- Il faut toujours tenir compte des utilisateurs du Web qui utilisent des accès à très faible taux de transfert : télécharger 500 ko = 8*500 = 4000kbit par page prend (environ) 4000/28.8 ~ 140 secondes, plus de deux minutes, avec une connexion à 28.8 kbit/s.
- Attention, seuls les navigateurs graphiques permettent de visualiser les images; il faut penser à ne pas priver les autres visiteurs du contenu tant que c'est possible

Format d'images pour le Web

- Différents formats d'images sont reconnus par les navigateurs:
  - les formats GIF et JPEG sont les deux formats qui sont utilisés depuis des années et ne souffrent d'aucun problème de compatibilité,
  - le format PNG est apparu plus tardivement, mais est très bien supporté aujourd'hui.
- Ces trois formats sont de type « images exprimées par leurs pixels »
- Il existe également des graphiques aux formats vectoriels : par exemple, le format SVG fondé sur XML
- Attention, dans les URL, la **casse** est importante; il sera bon de préférer écrire systématiquement nos extensions de fichiers en minuscules.

Extensions d'images: GIF, JPEG, PNG

Les extensions possibles, leur type MIME, leurs caractéristiques, et les utilisations indiquées de ces différents formats :

- Format : GIF Extension : .gif Type MIME : image/gif Le format GIF, pour ***Graphics Interchange Format***, est un format de ***haute compression d'images***
- Format : JPEG Extension : .jpg, .jpeg ou .jpe Type MIME : image/jpegJPEG, pour Joint Photographic Expert Group, est associé à un procédé de compression plus complexe que celui des images GIF; procédé également utilisé pour la compression vidéo MPEG.
  - Cette compression est **destructive** : il y a une perte de qualité qui augmente avec le taux de compression.
  - Le format ne gère pas la transparence, ni l'animation mais gère l'entrelacement et donc **l'affichage progressif** de l'image au fil du chargement, qui devient utile pour des images de taille plus importante.
- Format : PNG Extension : .png Type MIME : image/png Le format PNG signifie Portable Network Graphic. Spécialement conçu pour le Web par le W3C. Rassemble les avantages des deux formats précédents.

SVG

- Format : SVG Extension : .svg, .svgz Type MIME : image/svg+xml
  - Le format SVG (Scalable Vector Graphics) est un format d'image vectoriel et non de type bitmap comme les précédents.
  - Comme le PNG, c'est un format libre de droits, introduit par une spécification du W3C.
  - Il est fondé sur XML, ce qui implique que les données sont de type texte et non binaire : les éléments graphiques sont décrits (dimensions, couleur, position, remplissage des surfaces,...), structurés,...
  - Ils peuvent être **manipulés par des scripts** (modifications d'éléments graphiques,...).
  - On peut associer des liens aux éléments pour créer des images réactives.

SVG (Suite)

- On peut rendre ces images dynamiques (animées) et y associer du son.
- Aussi, la taille d'un fichier vectoriel est pratiquement systématiquement plus légère qu'un fichier bitmap.
- La résolution est une caractéristique absente : on peut agrandir une image vectorielle sans dégradation, pixellisation
- L'intégration d'images SVG se fait également à l'aide de la balise object en indiquant un message en cas d'erreur:

```
<object data="image.svg" type="image/svg+xml" width="200" height="200">
Image vectorielle au format SVG. Problème dans l'affichage. </object>
```

- On peut également utiliser directement la balise svg pour créer dans la page son image. Exemple:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
 <circle cx="100" cy="50" r="40" stroke="black"
        stroke-width="2" fill="red"/>
</svg>
```

Voir plus de détails ici : `https://developer.mozilla.org/fr/docs/Apprendre/HTML/Comment/Ajouter_des_images_vectorielles_%C3%A0_une_page_web`

La balise img

- La **balise <img />** est une **image** intégrée dans la page web xhtml (balise de type **EN LIGNE**), elle peut donc être intégrée dans le flot du texte. On ne peut pas rencontrer une image isolée dans le body.
- La balise <img /> doit obligatoirement contenir un attribut **src** (source) et **alt** (texte alternatif)
- *NB: <img /> ne peut pas être imbriquée dans une balise <pre>.*
- Utilisation :

```
<div>
   <img src="image/w3c.gif" alt="Logo W3C XHTML1.0 valide" />
</div>
```

affichera : 

(à condition, bien sûr, que l'arborescence du site contienne bien un répertoire « image » et le fichier gif en question dedans)

La balise <map>

- La **balise <map>** est une **image réactive** de la page web xhtml (balise de type **BLOC**).
- <map> regroupe les différentes zones réactives d'une même image. Elle regroupe donc les balises **<area />**.
- <map> possède obligatoirement un identifiant (attribut **id**).
- L'attribut **name** permet de faire référence à l'image sur laquelle on veut intégrer une ou plusieurs zones *réactives*. Attention, `name` est déprécié en XHTML. En HTML5, si les 2 attributs `name` et `id` sont utilisés, il doivent avoir la même valeur.
- *NB: la valeur de l'attribut id doit être identique à celle de l'attribut name.*
- Utilisation possible :

```
<img src="/image/w3c.gif" alt= "Logo du W3C" usemap="#logo_w3c" />
<map name= "map_logo_w3c" id= "logo_w3c">
    <area shape="rect" coords="0,0,31,31" href="http://www.w3c.org"
        alt="la partie gauche du logo du w3c" />
</map>
```

Images en tant que liens

- L'utilisation d'images en tant que lien : le contenu de la balise <a> peut être une image. On obtient donc le code :

```
<a href="../index.html" class="lien-navigation"
title="vers page d'accueil"> <img src="img/image.png"
class="image-lien" title= "Accueil" alt="Page d'accueil"
width="50" height="66" /> </a>
```

- L'image lien **s'intègre dans le flot** du texte, comme un lien classique, ou comme une image, à l'endroit où elle est intégrée dans le code.
- Si l'URL est incorrect ou si l'image est indisponible, a disparu, cela fera apparaître, à la place, le texte alternatif de l'image : il doit donc être choisi dans ce contexte d'image-lien, de manière à indiquer la nature du lien, comme on le faisait pour un contenu texte d'une balise `a`.
- Il ne faut pas hésiter à employer des `title` ; on peut observer que sur la bordure, c'est l'infobulle du lien qui s'affiche, sur l'image, c'est le titre de l'image. Les classes sont choisies pour rappeler le rôle des éléments.

HTML5 et XHTML

- Le choix de parler surtout de XHTML s'explique par le fait que HTML5 est plus laxiste que XHTML (<head> et <body> peuvent être implicites, abandon des notions de block (BLOC) et inline (EN LIGNE) en CSS...)
- Mais HTML5 est intéressant car il a été pensé pour être plus interactif avec ses APIs (<canvas>, <audio>, <video>, glisser-deposer...)
- `<canvas>` : balise qui définit une zone rectangulaire sur la page. Par défaut, un canvas n'a pas de bordure et pas de contenu.
  - Exemple:

    ```
    <canvas id="monCanvas" width="200" height="100"></canvas>
    ```
- `<audio>` : pour jouer un fichier audio
  - Exemple:

    ```
    <audio controls>
      <source src="horse.ogg" type="audio/ogg">
      <source src="horse.mp3" type="audio/mpeg">
        Your browser does not support the audio element.
    </audio>
    ```
- `<video>` : même chose pour la vidéo

HTML5

- Le tableau suivant récapitule les nouveaux éléments de section et leur usage le plus commun, tel que décrit par la spécification.

Nom	Détails
<section>	Section générique regroupant un même sujet, une même fonctionnalité, de préférence avec un en-tête, ou bien section d'application web
<article>	Section de contenu indépendante, pouvant être extraite individuellement du document ou syndiquée (flux RSS ou équivalent), sans pénaliser sa compréhension
<nav>	Section possédant des liens de navigation principaux (au sein du document ou vers d'autres pages)
<aside>	Section dont le contenu est un complément par rapport à ce qui l'entoure, qui n'est pas forcément en lien direct avec le contenu mais qui peut apporter des informations supplémentaires.
<header>	Section d'introduction d'un article, d'une autre section ou du document entier (en-tête de page).
<footer>	Section de conclusion d'une section ou d'un article, voire du document entier (pied de page).

HTML5 (suite)

- Il n'y a qu'une seule façon de déclarer un document HTML5 :
  - `<!DOCTYPE html>` (à mettre en toute première ligne du document)
- HTML5 **n'est pas fondé sur SGML**, contrairement à XHTML et à HTML 4.01, donc **ne nécessite pas de référence à une DTD** (comme on a pu le voir précédemment pour XHTML)
- HTML5 est compatible avec XHTML et autorise des documents XHTML5
- Ainsi, HTML5 spécifie deux syntaxes : HTML5 et XHTML5

XHTML5

- Ainsi, pour déclarer un document XHTML5, deux solutions sont possibles :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr"
    lang="fr">
```

ou bien

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>...</title>
    <meta charset="UTF-8" />
</head>
...
```

Nota Bene

- Les **attributs** dans les nœuds sont, en principe, **fixés**.
- Mais il est permis d'utiliser ses propres attributs :
  - Exemple : `<input type='radio' name='marque' data-lat='48.83' data-long='9.15' value='Porsche'>`
  - `data-lat` et `data-long` ne sont pas des attributs prédéfinis

- Ca peut être utile (on le verra dans le CM PHP), mais il faut faire attention, voir ici : http://w3c.github.io/html/single-page.html#embedding-custom-non-visible-data-with-the-data-attributes

CSS

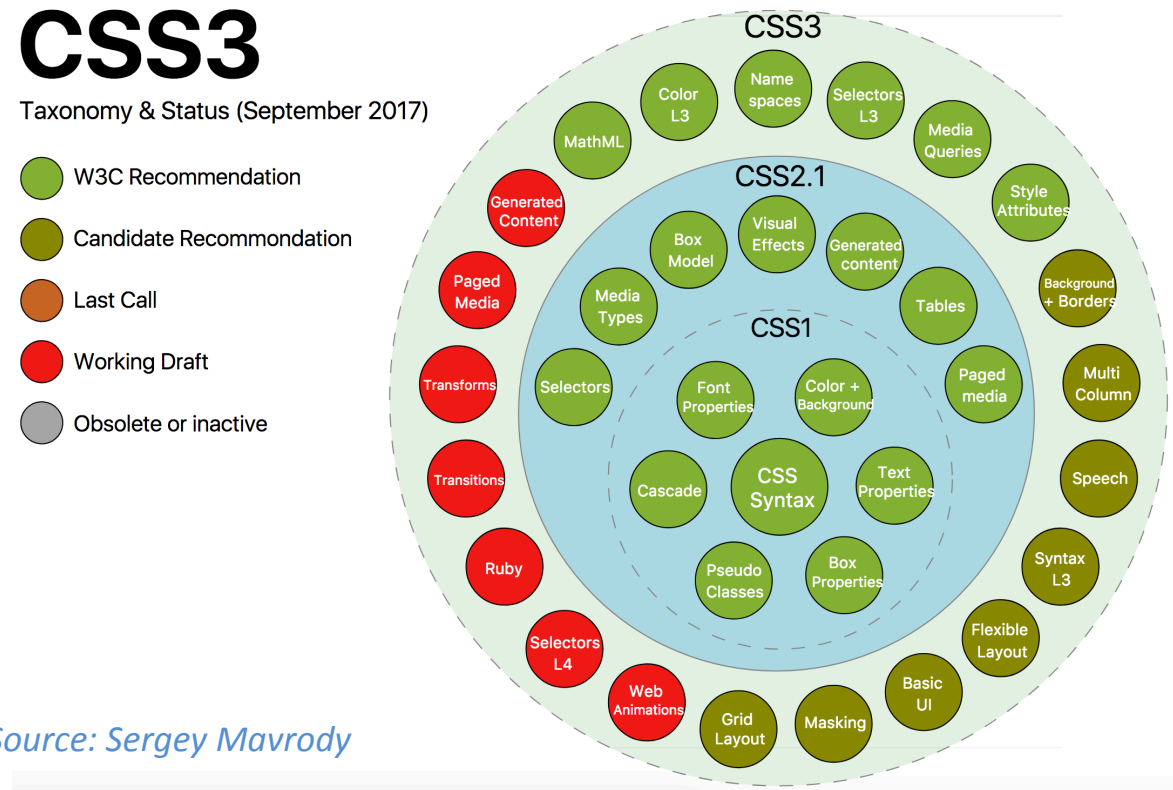
M2 G2M, Univ. Paris 8

par Isis TRUCK

Inspiré de diverses sources

Historique

- CSS : Cascading Style Sheets
- Version actuelle : CSS3, bientôt CSS4 (en cours de développement)
- Tous les navigateurs ne supportent pas encore les sélecteurs développés dans CSS4 : voir test ici : http://css4-selectors.com/browser-selector-test/



Source: Sergey Mavrody

CSS

- Une « feuille de style en cascade » est un ensemble de règles.
- Chaque règle comprend un sélecteur et un descripteur:
  - `selecteur {descripteur}`
  - le sélecteur désigne les éléments XHTML sur lequels s'appliquent les règles définies dans le descripteur
  - descripteur: suite de règles séparées par un `;`
  - Chaque règle est de la forme :
    `propriété : valeur;`

Que fait l'exemple ci-dessous ?

```
/* mystyle.css */
h1 {
     text-align : center; }
h2 {
     color : #0000FF; }
p {
     text-indent : 2em; }
p::first-letter {
     font-size : 120%; }
```

Appel du CSS depuis XHTML

- Déclaration dans la balise `<head>` :

```
<link href="adresse" type="text/css"
title="titre" rel="stylesheet"/>
```

- La balise `link` lie ce fichier XHTML à la ressource CSS dont l'`adresse` est, par exemple, l'URL relatif suivant:

```
css/screen.css
```

  - Ici, c'est une mise en page prévue pour l'affichage à l'*écran*, nous pourrions en définir une autre pour l'*impression* ou pour un autre type de média.

Déclaration du CSS directement dans XHTML

- Il est **possible**, mais **il vaut mieux l'éviter**, d'écrire le code CSS directement dans le fichier (X)HTML :
  - Utilisation de la balise `<style>` :

```
<style type="text/css">
      /* css */
</style>
```

- Il vaut mieux l'éviter pour cause de **lisibilité** et de **réutilisation** du code CSS (pour d'autres fichiers (X)HTML)

Autres éléments importants

- Déclaration de règles pour une classe en CSS :

```
element.nom                  /* element = p, table, td, span, ... */
{                            /* nom = nom de la classe (au choix) */
   propriete: valeur;
   propriete: valeur;
   ...
}
```

- Pour déclarer des règles pour une classe valide pour tous les éléments:

```
.nom
{
   propriete: valeur;
   propriete: valeur;
   ...
}
```

- Les valeurs par défaut (pas de classe):

```
element
{
   propriete: valeur;
   propriete: valeur;
   ...
}
```

Autres éléments importants (2)

- Déclaration de règles pour un `id` en CSS :

```
element#nom-id                /* element = p, table, td, span, ... */
{                             /* nom-id = id (au choix) */
   propriete: valeur;
   propriete: valeur;
   ...
}
```

- Déclaration de règles pour le nœud fils d'un objet identifié par un `id` :

```
p#nom-id a {
   background-color: red;
}
```

Extrait du code HTML associé :

```
...
<p id="nom-id">
    <a href="http://www.w3c.org">Lien vers le site du W3C</a>
</p>
...
```

Autres éléments importants (3)

- Déclaration de règles communes à plusieurs éléments : on indique les éléments à la suite, en les séparant par des **virgules** :

```
element1, element2, element3  {
   propriete: valeur;
   propriete: valeur;
   ...
}
```

- Exemple :

```
p, h1, h3  {
   background-color: yellow;
   color: red;
}
```

Autres éléments importants (4)

Pseudo-classes et pseudo-éléments

- Pseudo-classe : notation **:**
  - C'est un mot-clef, précédé d'un 'deux-points', ajouté après un sélecteur pour spécifier que l'on souhaite donner un style à l'élément sélectionné lorsqu'il est dans un certain état.
  - Par ex., on peut vouloir styler un élément seulement lorsqu'il est survolé par la souris, ou bien styler un élément qui est le 1er fils de son parent dans le DOM.
  - Exemples:
    - `:active`
    - `:checked`
    - `:nth-child()`
    - `:first`
    - `:hover`

Autres éléments importants (5)

Pseudo-elements ::

- Ils ressemblent aux pseudo-classes. Ils sont également des mots-clefs, précédés par un double 'deux-points', qui peuvent être ajoutés à la fin d'un sélecteur pour sélectionner une **certaine partie** d'un élément.
- Exemples:
  - `::after`
  - `::before`
  - `::first-letter`
  - `::first-line`
  - `::selection`
  - `::backdrop` (cf. `https://developer.mozilla.org/fr/docs/Web/CSS/::backdrop`)

- La spécification CSS3 préfixe les pseudo-éléments avec un double deux-points au lieu d'un seul. Donc `:first-letter` devient `::first-letter`, par exemple. IE 8 et les versions antérieures ne supportent pas le préfixe double deux-points.

Bootstrap

- ensemble d'outils *open source* (*framework* d'interface) pour développer en HTML, CSS, JS.
- Consiste en un ensemble de feuilles de style CSS écrites en **Less**
- Less :
  - est un préprocesseur CSS (qui génère dynamiquement du CSS) permettant d'utiliser des variables, *mixins*, etc.
  - Ex. variables :

```
@color: #4D926F;
#header {
  color: @color;
}
h2 {
  color: @color;
}
```

Bootstrap

- Ex. mixin (« appeler » dans une classe des propriétés définies dans une autre classe) :

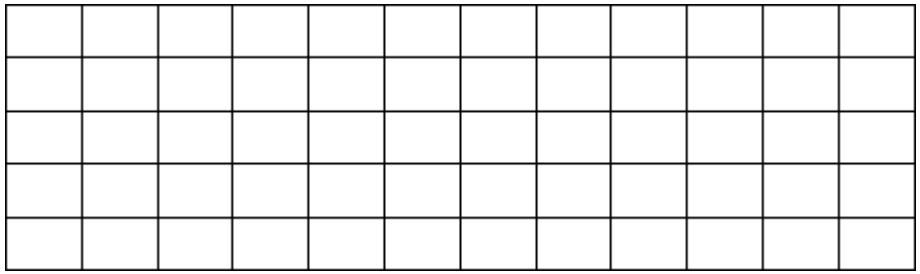
```
.bordered {
  border-top: dotted 1px black;
  border-bottom: solid 2px black;
}

.post a {
  color: red;
  .bordered;
}
```

- Fonctionnement :
  - Une feuille de style principale (appelée `bootstrap.less`) englobe les feuilles de style des composants
  - Les développeurs choisissent les composants qu'ils souhaitent en modifiant la feuille principale
  - Bootstrap a un système de **grille** pour agencer l'aspect visuel de la page web

Bootstrap : grille

- découpage en cellules (row, col) avec 12 colonnes :

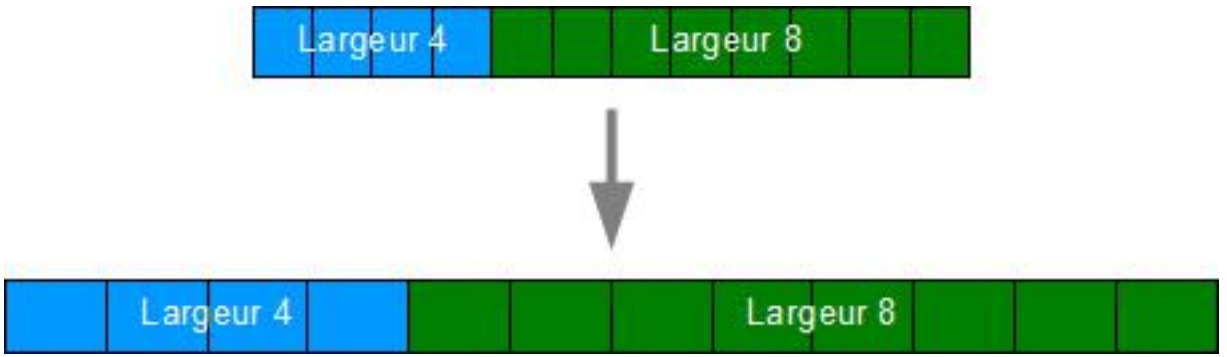


- On organise le contenu en utilisant pour chaque élément une ou plusieurs cellules :

Source : https://openclassrooms.com/courses/prenez-en-main-bootstrap/une-grille

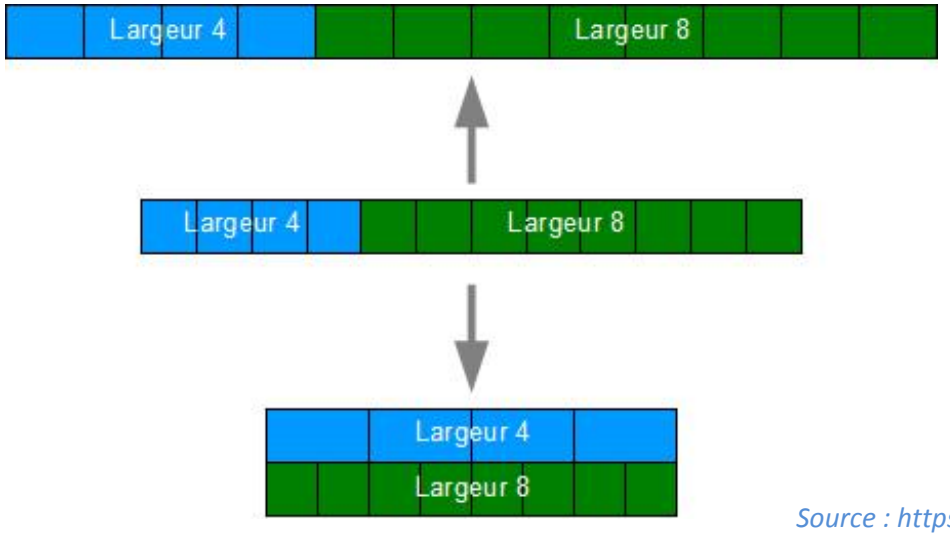
Bootstrap : les *media queries*

- Elles permettent de cibler le *device* du client :
  - Ordinateur
  - Tablette
  - Smartphone
  - …
- pour mieux gérer l'affichage de la page selon la résolution, l'orientation, la taille de l'écran, etc. => ***responsive design***



Source : https://openclassrooms.com/courses/prenez-en-main-bootstrap/une-grille

Bootstrap : les *media queries*



Source : https://openclassrooms.com/courses/prenez-en-main-bootstrap/une-grille

- Elles utilisent la règle `@media`
  - Ex: Si la fenêtre du navigateur a une taille inférieure à 500 pixels, alors le fond d'écran devient bleu clair :

```
@media only screen and (max-width: 500px) {
    body {
        background-color: lightblue;
    }
}
```

Source : https://www.w3schools.com/css/css_rwd_mediaqueries.asp