

HTML



CSS



JS



Techniques et Langages du Web

GREGORY MOREL - CPE 2022-2023 – 3ETI /3IRC

Equipe pédagogique

Enseignants

Grégory Morel

Ens.-Chercheur CPE

Graphes, algorithmes et
optimisation

Respo. Majeure Info

gregory.morel@cpe.fr

Bureau : B127A



John Samuel

Ens.-Chercheur CPE

Ex Yahoo
Analyse de données

john.samuel@cpe.fr

Bureau : B127A

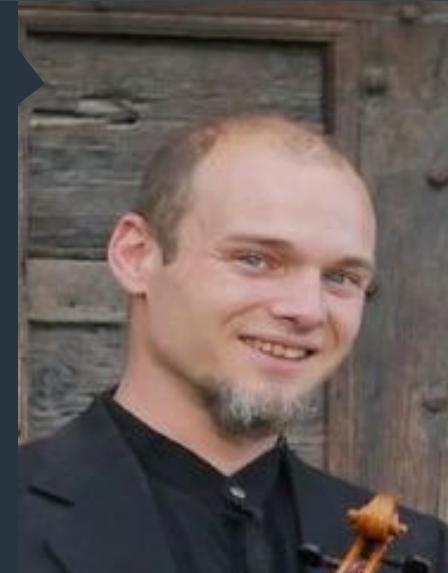


Bruno Mascret

Ens.-Chercheur CPE

Spécialiste des nouvelles
technologies pour le
handicap

bruno.mascret@cpe.fr



Equipe pédagogique

ON N'A PRIS QUE LES MEILLEURS !

Intervenants TP / Projets

Pierre Avinain

CPE ETI 2021

Développeur auto-entrepreneur et en entreprise

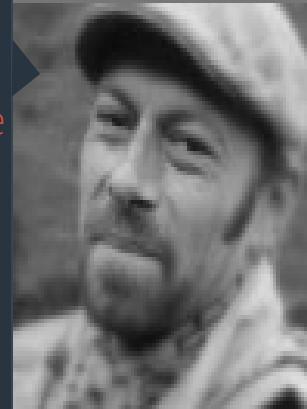
Pixel Perfectionist



Alain Becker

13 ans d'exp. dans le développement

Intervient depuis 9 ans à CPE



Yannick Joly

CPE IRC 2010

4 ans resp. projet & développement

Enseignant CPE Lyon depuis 2021



Rubiela Carillo Rozo

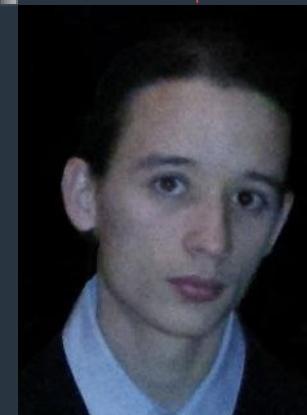
Docteure en Informatique (IHM)

Master en Socio-Anthropologie

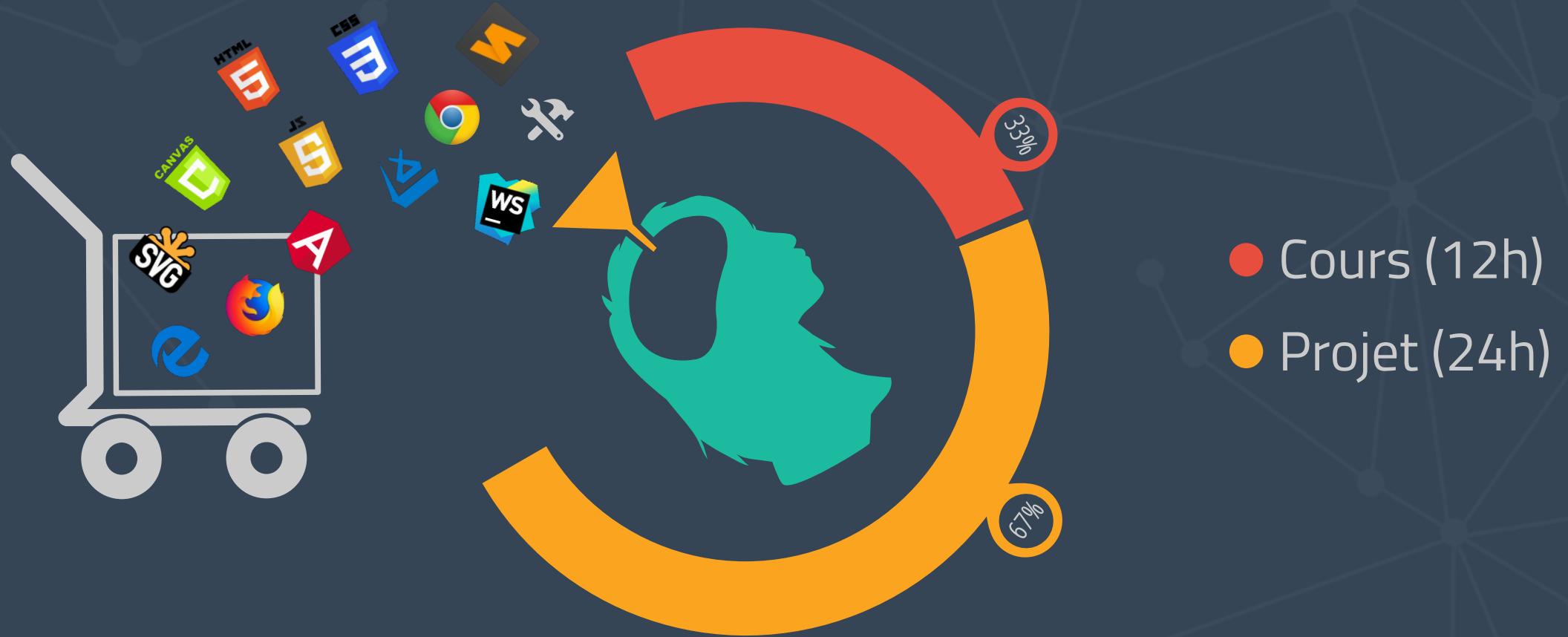


Alexis Kleinbauer

Ingénieur



Organisation du module ETI



Plan du cours



Evaluation



Examen sur machine

50%

Projet

50%

Implication, comportement...

variable

Ressources



De nombreux exemples de ce cours proviennent des sites

www.w3schools.com, developer.mozilla.org/fr/ (MDN)

et

www.alsacreations.com, <https://css-tricks.com/>

Il y a une infinité de ressources intéressantes sur le web !

Faites vos recherches en anglais, pensez à www.stackoverflow.com

Livres « aide-mémoire » gratuits : books.goalkicker.com/



Introduction : qu'est-ce que le web ?

Problématique



Un peu d'histoire

- 1969 : une équipe de recherche d'IBM crée **GML** (*Generalized Markup Language*)
Objectif : *annoter* les différents éléments d'un texte (titre, paragraphe...), afin de séparer le *fond* et la *forme* des documents, et avoir une représentation *indépendante* du système utilisé (est-ce qu'on visualise le document sur une écran ou imprimé ? En couleur ou en noir & blanc ?)

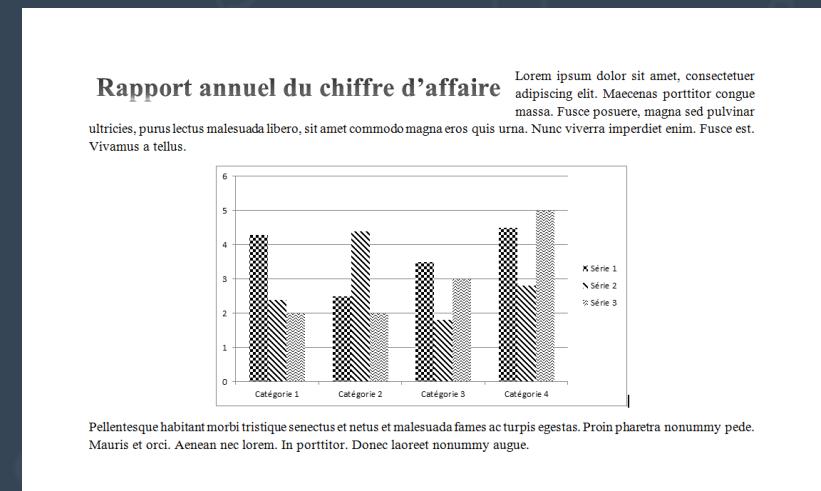
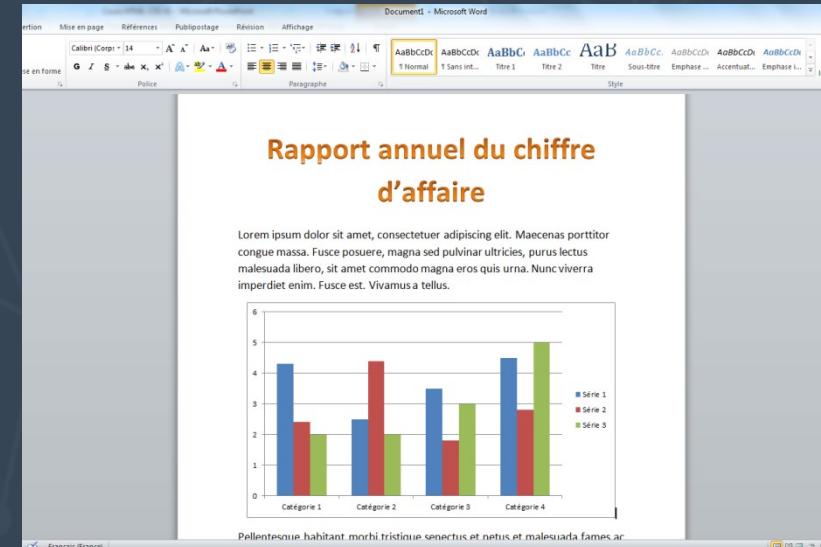
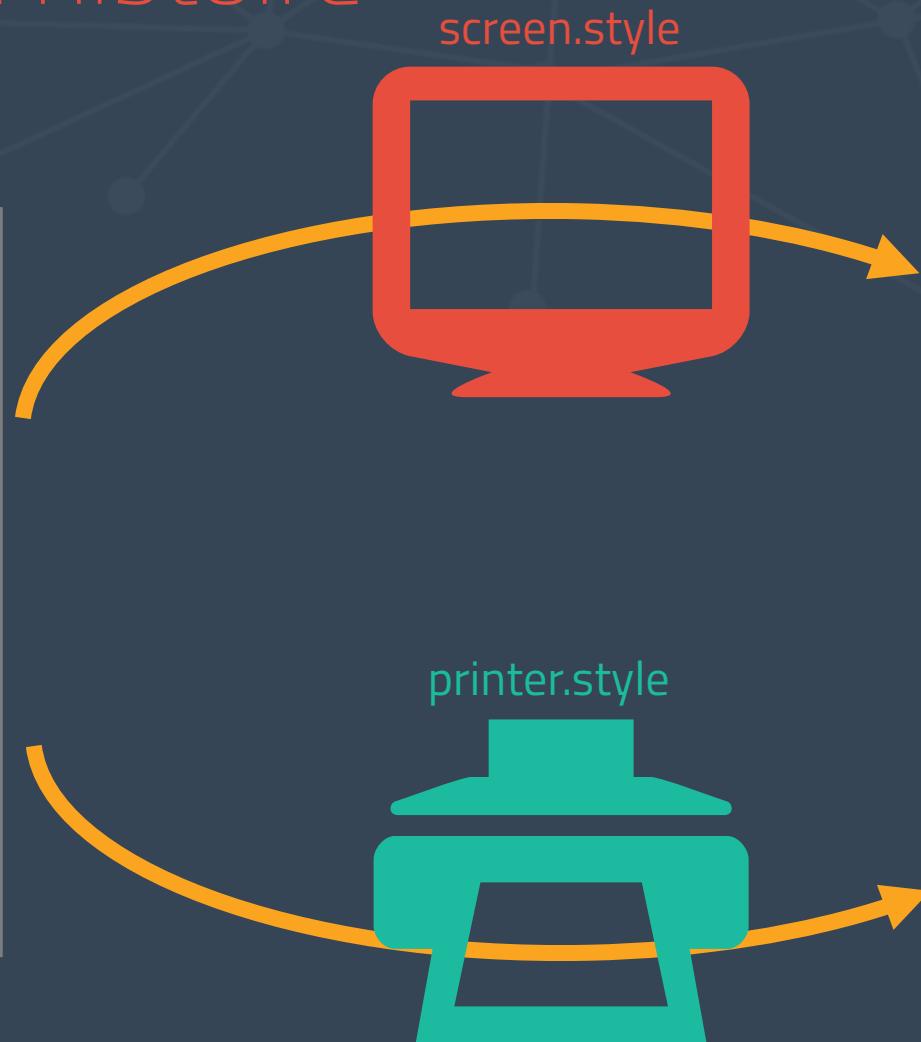
Avantages :

- donner une *sémantique* aux éléments : pour dire que "**Chapitre 1**" est le tire d'un chapitre, on ne l'écrit pas avec une police de taille 40, on lui ajoute une étiquette ou *balise* du type `<chapitre>`
- permet de *changer facilement le style* d'un document : par exemple la police, la taille, la couleur des intitulés de chapitre

Un peu d'histoire

```
<Title> Rapport annuel  
du chiffre  
d'affaire</Title>  
  
<Intro>Lorem  
Ipsum...</Intro>  
  
<Graph>resultats.png  
</Graph>  
  
<Comments>Pellentesque  
habitant</Comments>
```

rapport.gml



Un peu d'histoire

- Années 70 – 80 : apparition des *DTD* (Document Type Definition), qui décrivent la *structure* d'un document GML et permettent de le *valider*:

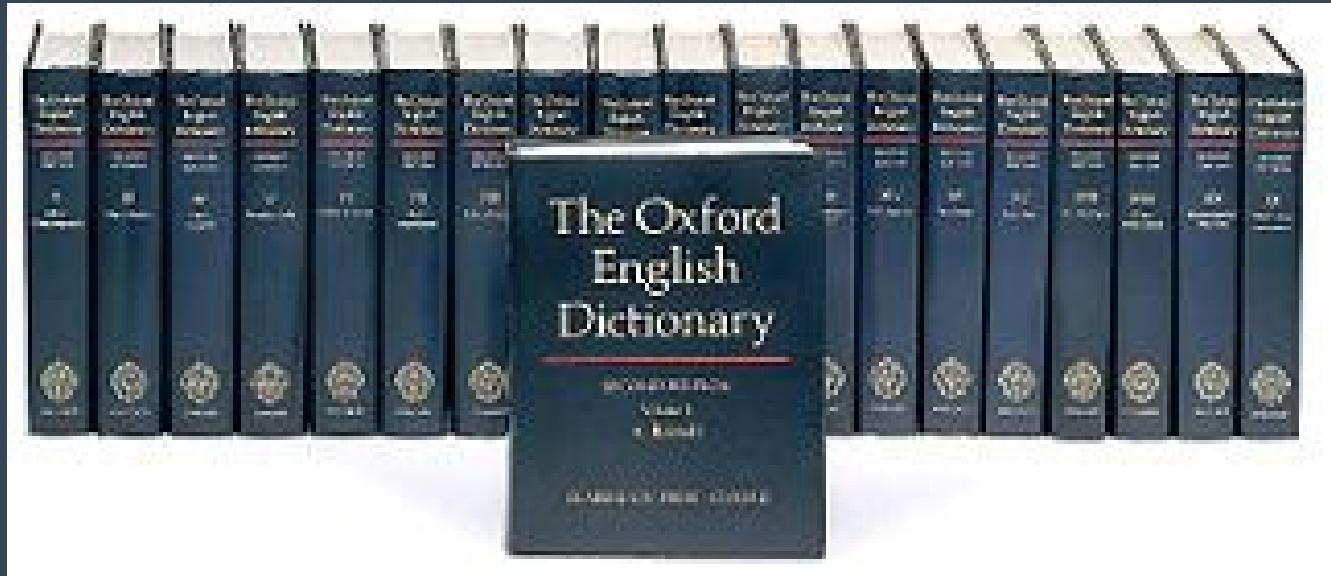
| DTD pour un document de type <i>Note à un collègue</i> | Exemple de document conforme |
|--|---|
| <pre><!DOCTYPE note [<!ELEMENT note (to,from,heading,body)> <!ELEMENT to (#PCDATA)> <!ELEMENT from (#PCDATA)> <!ELEMENT heading (#PCDATA)> <!ELEMENT body (#PCDATA)> >]</pre> | <pre><note> <to>Tove</to> <from>Jani</from> <heading>Reminder</heading> <body>Don't forget me this weekend</body> </note></pre> |

- 1984 : GML devient un standard sous le nom *SGML* (Standard GML)

Un peu d'histoire

- Années 80 : grand succès de SGML

L'OED (Oxford English Dictionary), le plus gros dictionnaire au monde (20 volumes, 21 728 pages) est entièrement composé en SGML !



Document: Bungler OED At: "<entry>"

```
<entry>
  <hwsec>
    <hwsp>
      <hwlem>bungler</hwlem>
      <pron>b<i>ʌŋglaɪə</i></pron>. </hwsp>
    <vfl>Also <vd>b</vd> <vf>bongler</vf>,
      </vfl>
    <etym>f. as prec. + <xra><xlem>-ER</xlem></xra>
  <sen>One who bungles; a clumsy unskillful person.
  <quot>
    <qdat>1593 </qdat>
    <auth>MORE </auth>
    <wk>Answe. Poyson. Bk. </wk>Wks. (1557)
    <qtxt>He is even but a very bungler.
```



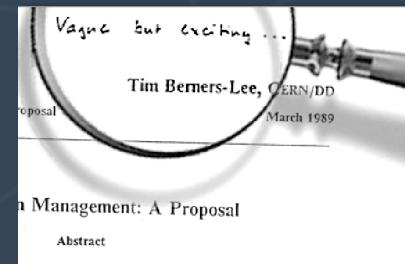
Un peu d'histoire

- 1984 : projet CERNDOC au CERN (en SGML) : réorganiser toute la documentation interne

Tim Berners-Lee n'est pas satisfait par CERNDOC et travaille sur un projet personnel de son côté, en exploitant notamment la notion de *lien hypertexte*

- 1989 : Berners-Lee écrit un mémo sur son idée : *Information Management: A Proposal*

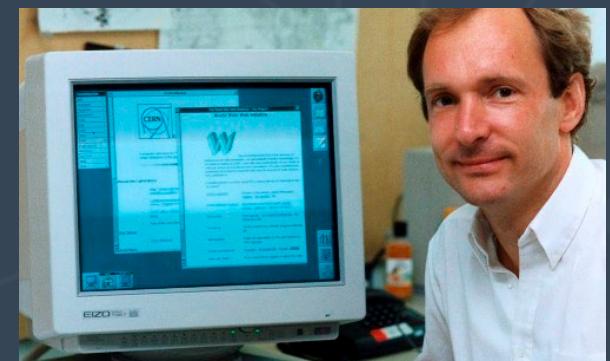
Réaction de son chef : « *Vague, but exciting...* »



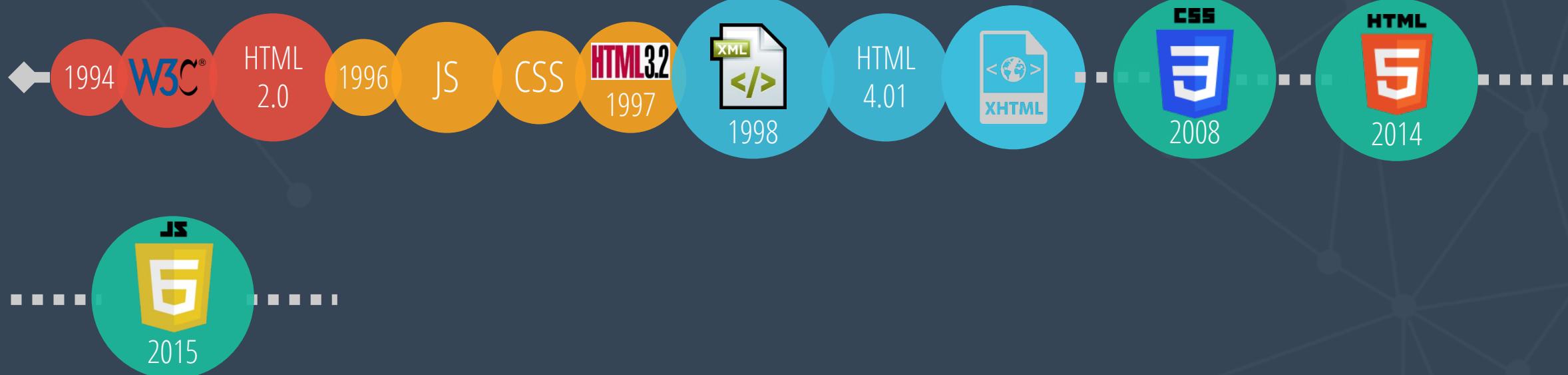
- 1990 : Berners-Lee travaille sur une application, plus simple, de SGML : HTML (*HyperText Markup Language*)

Nouvelle publication : *WorldWideWeb, proposal for a hypertext project*

Création de HTTP, du premier navigateur



Un peu d'histoire



Un peu d'histoire

Années 1990



<https://archive.org/web/>

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents. Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#), [Policy](#), November's [W3 news](#), [Frequently Asked Questions](#), [What's out there?](#) Pointers to the world's online information, [subjects](#), [W3 servers](#), etc. [Help](#) on the browser you are using. [Software Products](#) A list of W3 project components and their current state. (e.g. [Line Mode](#), [X11 Viola](#), [NeXTStep](#), [Servers](#), [Tools](#), [Mail robot](#), [Library](#)) [Technical](#) Details of protocols, formats, program internals etc. [Bibliography](#) Paper documentation on W3 and references. [People](#) A list of some people involved in the project. [History](#) A summary of the history of the project. [How can I help?](#) If you would like to support the web. [Getting code](#) Getting the code by [anonymous FTP](#), etc.

Welcome to Apple 1997

Introducing CyberDrive
Register today for a free CD-ROM.

Preorder Mac OS 8
Now you can [preorder Mac OS 8](#), described by Steve Jobs as "the most revolutionary update to the Mac OS in years," sporting a bold new look, a speedier Finder, more sheetstacking, integrated Internet functions.

Want a PowerBook?
Qualify to win a [PowerBook 2400c](#) by entering this month's Apple Registration Sweepstakes.

Big Help for Small Biz Find out what you need to make your small business a success. [Apple Business](#) has all the information about everything from great deals on leasing to dealing with the SBA.

TECHNIQUES ET LANGAGES DU WEB – 2022-2023



Welcome to Amazon.com Books!

*One million titles,
consistently low prices.*

(If you explore just one thing, make it our personal notification service. We think it's very cool!)

SPOTLIGHT! -- AUGUST 16TH

These are the books we love, offered at Amazon.com low prices. The spotlight moves **EVERY day** so please come often.

ONE MILLION TITLES

Search Amazon.com's [million title catalog](#) by author, subject, title, keyword, and more... Or take a look at the [books we recommend](#) in over 20 categories... Check out our [customer reviews](#) and

Google Search Engine

This is a demo of the Google Search Engine. Note, it is research in progress so expect some downtime and malfunctions. You can find the older [Backrub web page here](#).

Google is being developed by [Larry Page](#) and [Sergey Brin](#) with very talented implementation help by [Scott Hassan](#) and [Alan Steremberg](#).



Search Stanford

10 results clustering on Search

Search The Web

10 results clustering on Search

Un peu d'histoire

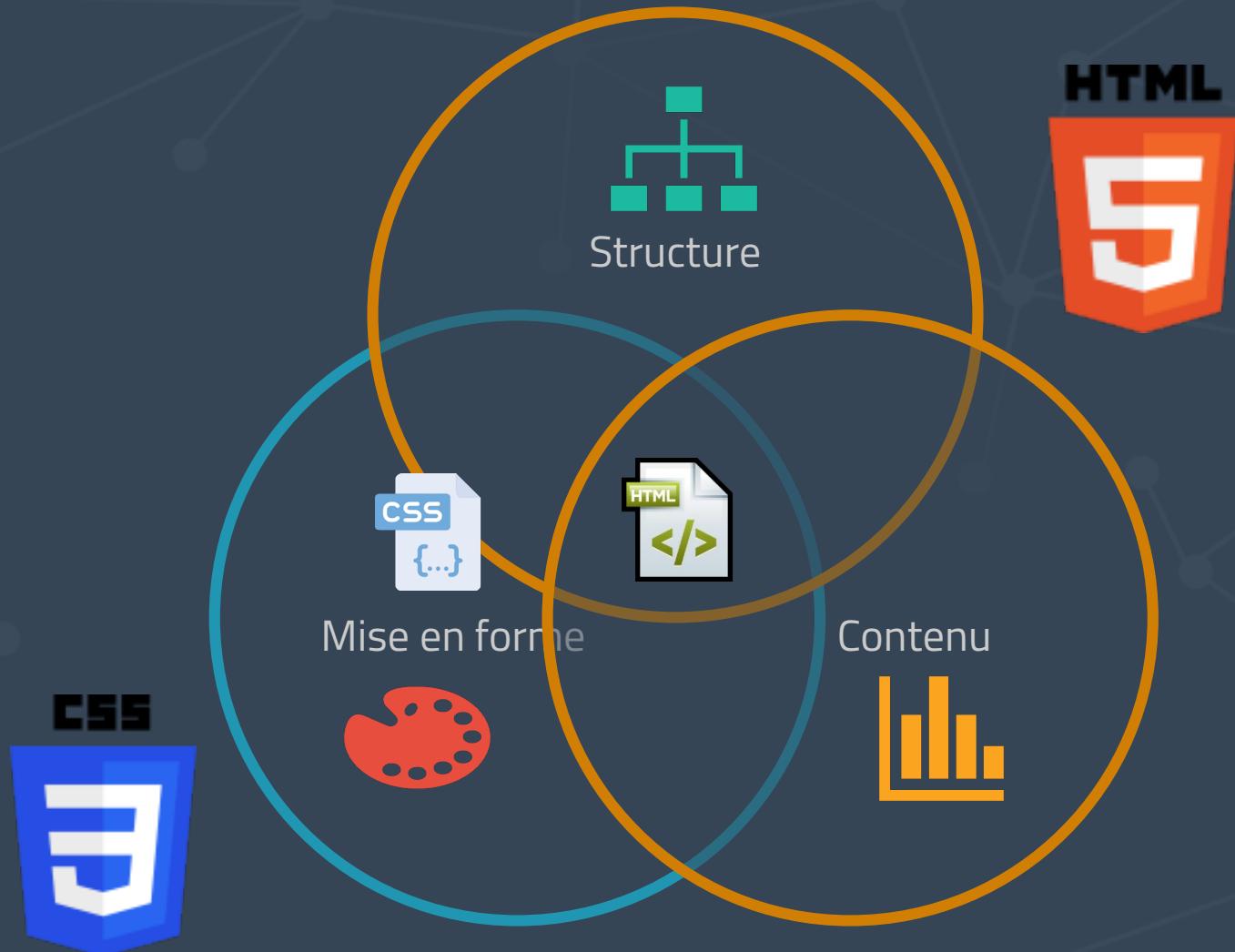
2005



2019

<https://archive.org/web/>

HTML vs CSS



Client / Serveur - HTTP

Requête : GET cpe.fr/index.html HTTP/1.1



<https://cpe.fr/index.html>

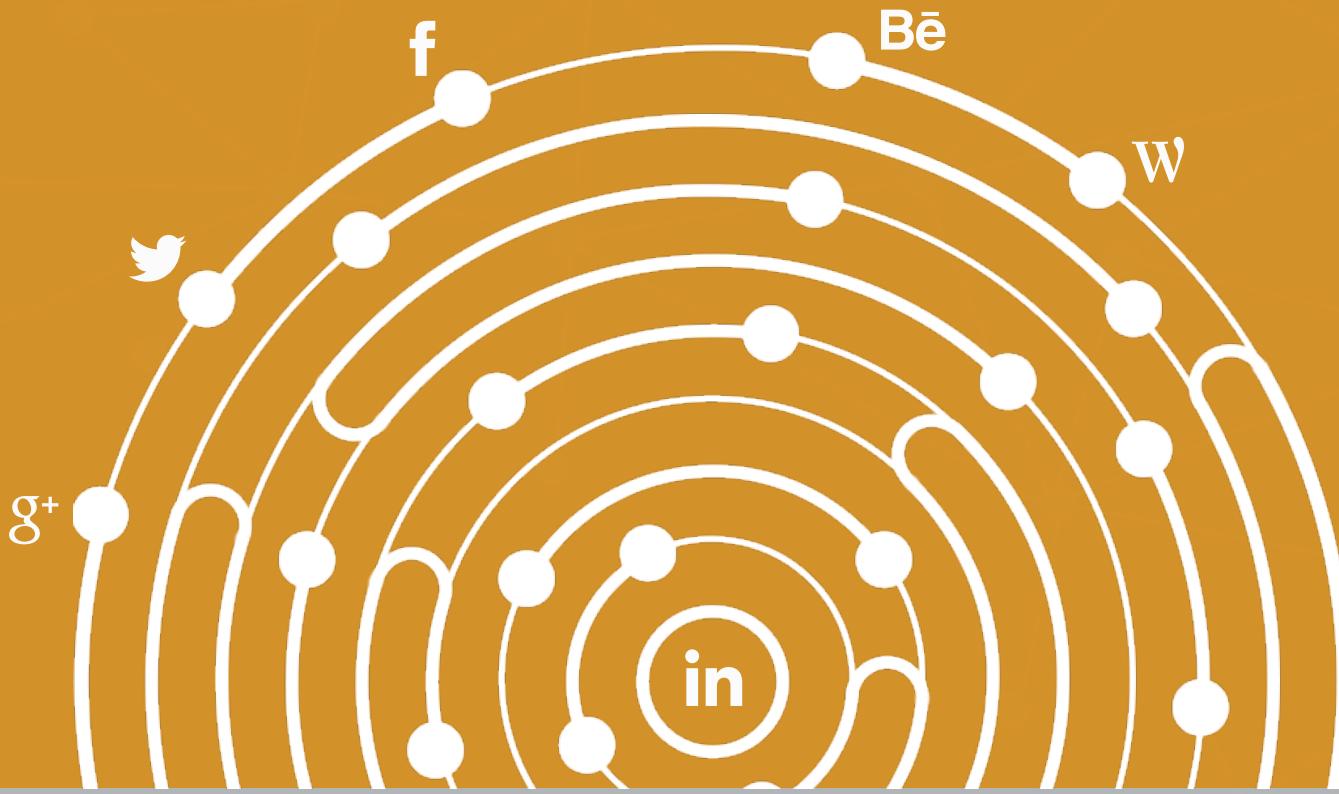
Réponse : status 200

OK ! Voici index.html



Client / Serveur - HTTP

- HTTP(S) = HyperText Transfer Protocol (Secure)
- Méthodes (ou *verbes*) : GET, POST, PUT, DELETE (+ HEAD, PATCH, CONNECT, TRACE, OPTIONS)
- En-tête :
 - de requête : method, accept, content-type, user-agent...
 - de réponse : status, content-type, content-length, server...
- Codes réponses (statuts) (cf. https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP)
 - 1xx : informations
 - 2xx : succès (200 = OK, 201 = Crée)
 - 3xx : redirection (301 / 302 = déplacé de façon permanente / temporaire)
 - 4xx : erreur du client (403 = non autorisé, 404 = ressource non trouvée)
 - 5xx : erreur du serveur (500 = erreur interne, 503 = service indisponible)



Projet & Méthodologie de conception

Projet

Sujet : Réaliser le site web d'un client proposant *plusieurs déclinaisons d'un produit à personnaliser*

Idées de produits : t-shirts, mugs, masques anti Covid, pizzas, caleçons...

Idées de déclinaisons : t-shirts avec ou sans manches, caleçon / slip / boxer...

Personnalisation : une image, un ingrédient, un message personnel...

⚠ Concentrez-vous sur **UN SEUL** type de produit, **SIMPLE**

⚠ Respectez le cahier des charges et les contraintes techniques imposés par le client !

Projet en binômes

Deadline : Dimanche 6 novembre 23:59:59

Phase 1 : analyse et définition

Identifier le contexte :

- **acteurs** du projet, dont le public visé
- existence d'une **charte graphique** ?
- **accessibilité** des autres pages web si poursuite d'un projet existant

Identifier les objectifs :

- quels sont les objectifs ? Visibilité, information, marketing
- quelle plateforme (desktop, mobile...) ?
- définition de ses **contraintes** et du niveau d'accessibilité

Identifier les moyens

- **qui fait quoi** ?
- en **combien de temps** ?
- **qui teste** ?



Phase 2 : développement de l'architecture

Collecter, préparer et organiser les ressources

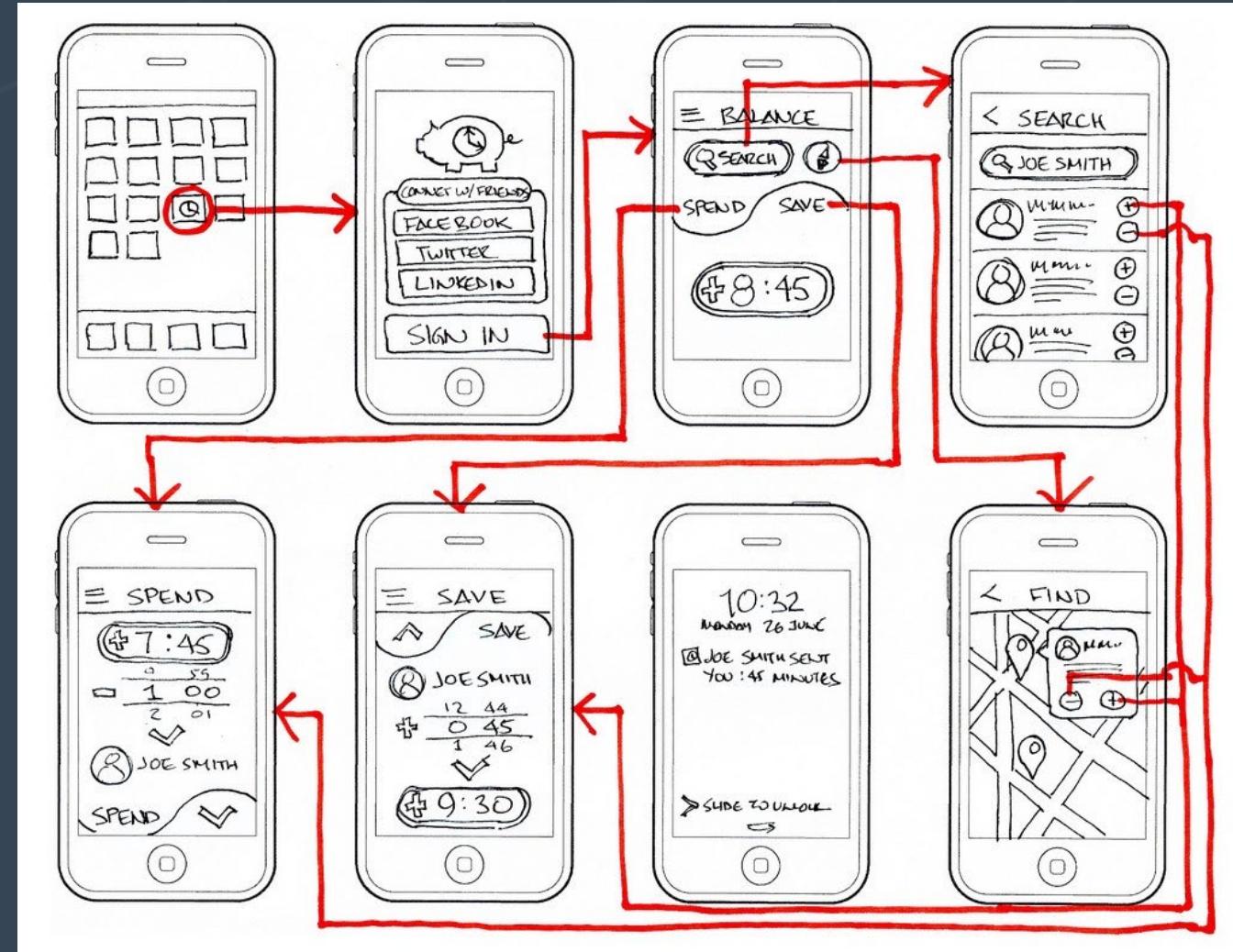
- percevoir l'organisation physique du site (hiérarchie, répertoires dédiés)
- une structure claire (et logique !) est facilement maintenable
- alternative aux formats propriétaires (conversion si besoin)
- attention aux ressources non accessibles !

Réaliser des patrons (*wireframes, mockups*)

- c'est là que les designers dans l'âme peuvent s'exprimer !
- penser aux ergonomes
- valider !

Identifier et séparer le contenu et la forme

Phase 2 : développement de l'architecture



Phase 3 : contenu

Elaborer le contenu

- définir les rubriques du site
- écrire les textes et messages d'impacts
- rassembler les illustrations (attention au droit d'auteur)

Saisir le contenu

- Regroupez les contenus par type (images, média...) ; à chacun son dossier !
- N'oubliez pas les **commentaires** (surtout en CSS et JavaScript)

Adapter

- Pensez **accessibilité** !

Phase 4 : mise en forme

Conception graphique

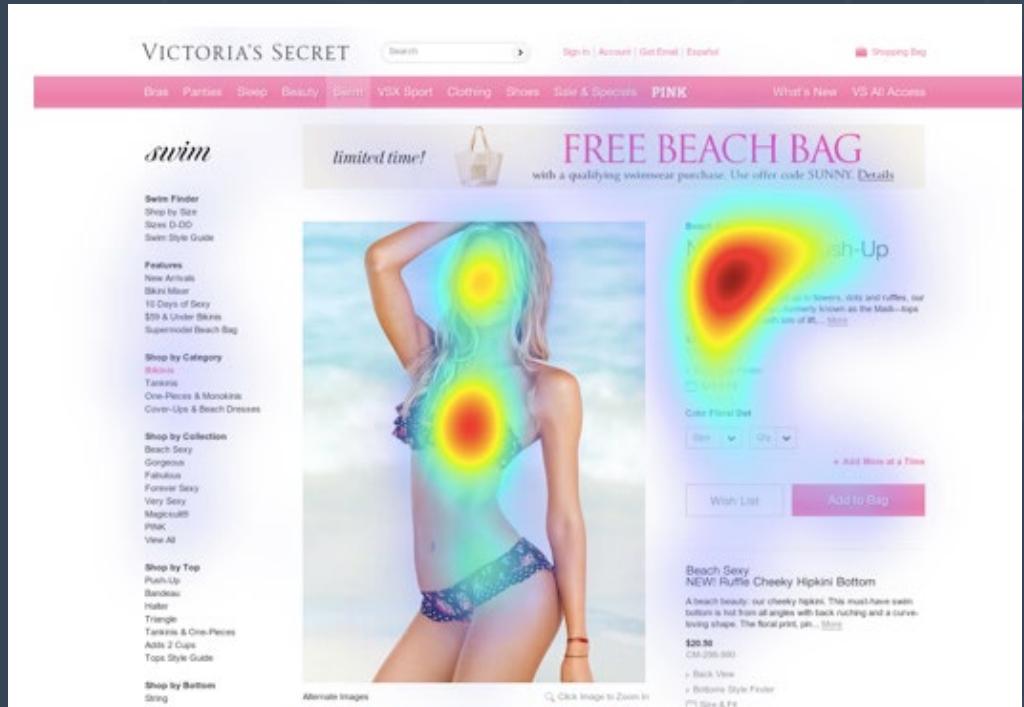
- créer la charte graphique
- créer l'interface de la page d'accueil et des pages « types »

Feuilles de style :

- directement liée à la structure du document
- elle est codée indépendamment
- il peut y en avoir plusieurs pour un même site.

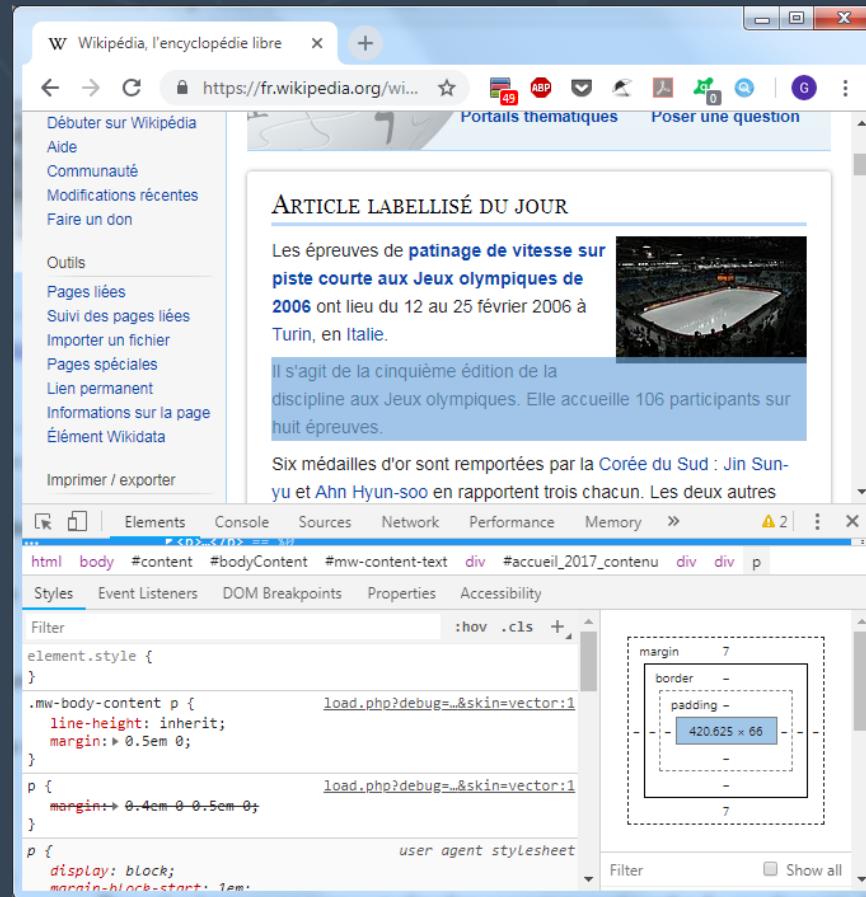
il ne devrait PAS y avoir de style dans des balises de contenu

Penser à l'"eye-tracking" (oculométrie)

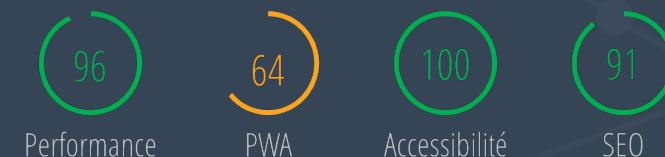


Phase 5 : tests et déploiement

Pensez aux **outils de développement** présents dans les navigateurs !



- Affichage et inspection du code source
- Débogueur, modification à la volée
- Tests de rendu sur téléphone, tablette...
- Console JavaScript
- Outils d'analyse réseau
- Tests de sécurité
- Audits



- Validateurs en ligne

Boîte à outils

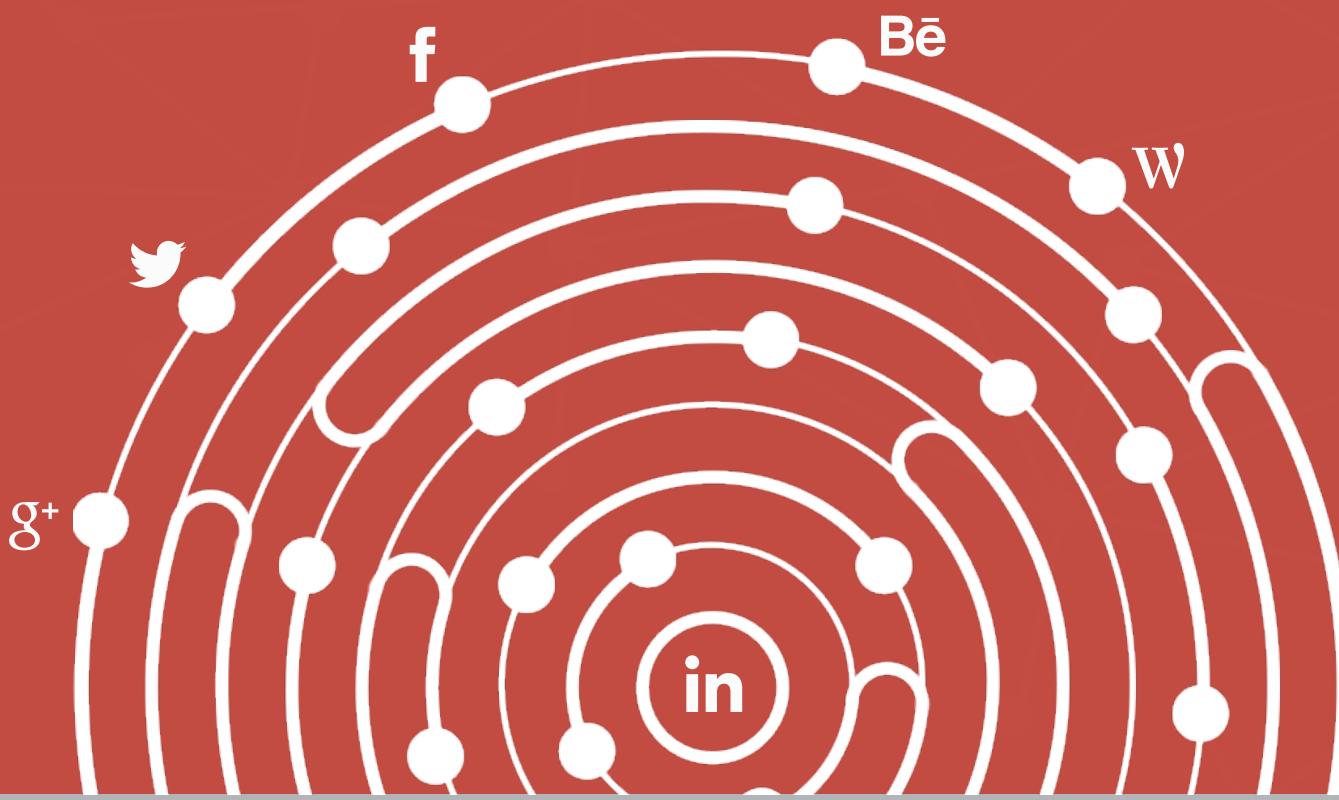
Pensez à utiliser les outils disponibles !

- <https://html-css-js.com/html/generator/>
- <https://divtable.com/generator/>
- <https://divtable.com/table-styler/>

Extension *Web Developer* (disponible pour Chrome et Firefox) ; permet

- de désactiver les feuilles de style (en totalité ou en partie), le code JavaScript...
- de compléter / modifier le comportement des formulaires
- de masquer les images / images de fond
- d'afficher des informations normalement « invisibles » (ancres, classes...)
- et surtout de valider votre code HTML, CSS ou l'accessibilité !

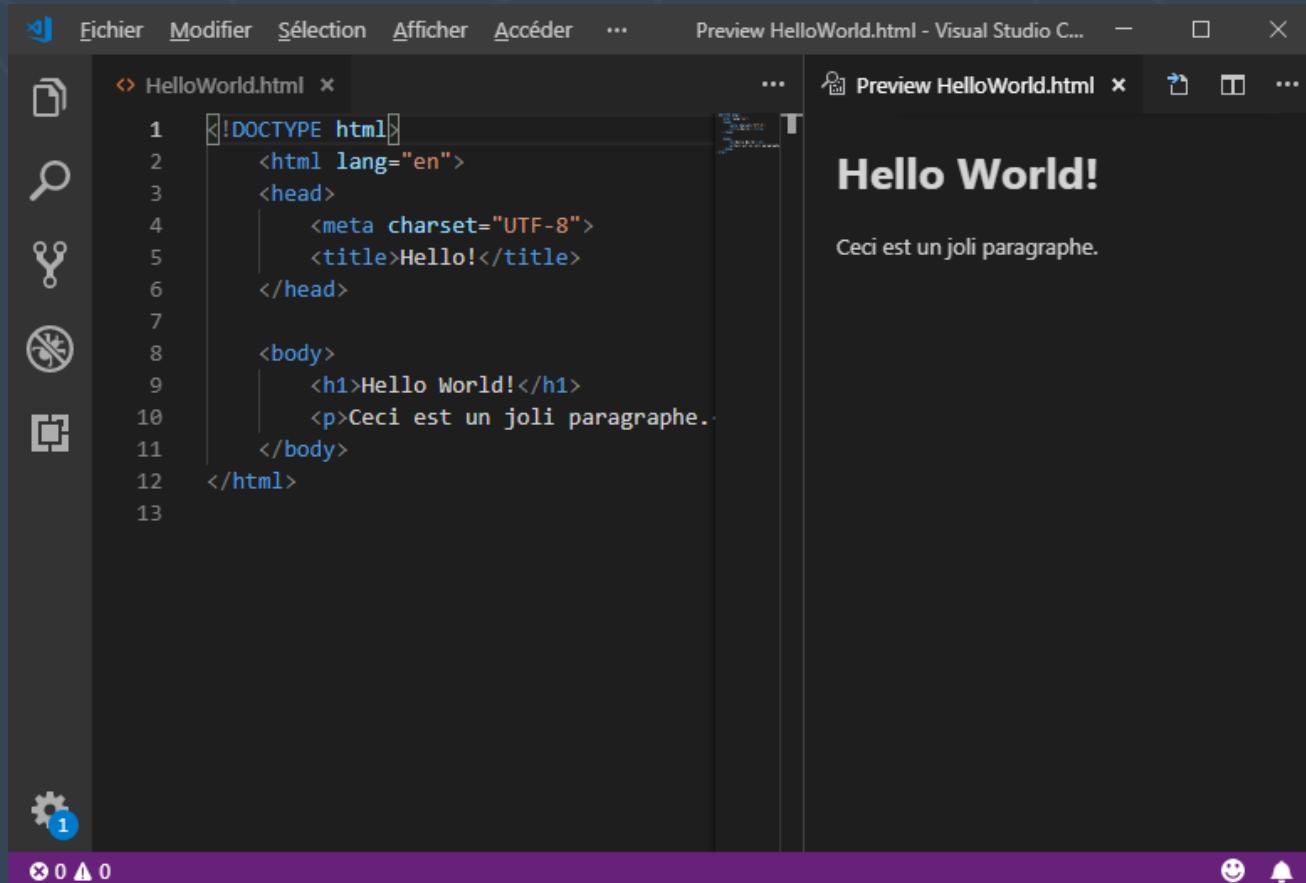
Essayez les exemples du site w3schools !



1^{ère} partie : HTML

Comment faire de l'HTML

CHOISIR UN BON EDITEUR



The screenshot shows the Visual Studio Code interface. On the left, the code editor displays the following HTML code:

```
1 <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="UTF-8">
5       <title>Hello!</title>
6     </head>
7
8     <body>
9       <h1>Hello World!</h1>
10      <p>Ceci est un joli paragraphe.</p>
11    </body>
12  </html>
```

To the right of the code editor is a preview window titled "Preview HelloWorld.html" which shows the rendered HTML content:

Hello World!
Ceci est un joli paragraphe.

- Le plus **basique**

bloc-notes + navigateur

- Plus **évolué** et pratique

VS Code



Sublime



Atom



Webstorm



- En ligne :

jsfiddle.net, codepen.io

html-online.com

💡 Extension **.html**

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8">
    <title>Hello!</title>
  </head>

  <body>
    <h1>Hello World!</h1>
    <p>Ceci est un joli paragraphe.</p>
  </body>
</html>
```

- Langage **balisé** (HyperText Markup Language)
- Généralement **1 balise ouvrante** et **1 balise fermante**

Ex. : `<p>` et `</p>`

- Parfois juste une balise
- Ex : ``
- Balises en **minuscules**

- `<!DOCTYPE html>`
- `<html>`
- `<head>`
- `<title>`
- `<body>`
- `<h1>`
- `<p>`

document HTML5
racine d'un document HTML
méta-information
titre du document
partie **visible** ou **audible** du document
en-tête de premier niveau
définit un paragraphe

Attributs

LES "PARAMETRES" D'UNE BALISE

- Fournissent des **informations supplémentaires** sur un élément de la page
- Toujours renseignés dans la balise **ouvrante**
- Généralement sous la forme **nom = "valeur"**
- Possibles pour **toutes les balises** HTML

Exemple :

```
<p lang="fr">Ceci est un paragraphe.</p>
```

Applications :

```
<p><q>Citation anglaise</q></p>
```

"Citation anglaise"

```
<p lang="fr"><q> Citation française </q></p>
```

« Citation française »

Synthèse vocale avec le bon accent !

Bases de l'HTML

EN-TÊTES : BALISES <hx>

- Balises d'en-têtes ; décrivent le contenu qui suit
- Définissent la **structure** du document
 - ➔ Utilisé par les **robots** pour l'indexation, par les **navigateurs** pour la mise en page...
 - ➔ Penser à la génération automatique de **table des matières**
- 6 niveaux : <h1> à <h6>

The screenshot shows a code editor on the left and a browser preview on the right. The code editor displays the following HTML structure:

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Hello!</title>
6   </head>
7
8   <body>
9     <h1>Main title</h1>
10    <p>Introduction</p>
11
12    <h2>Reasons</h2>
13      <h3>Reason 1</h3>
14      <p>Paragraph</p>
15
16      <h3>Reason 2</h3>
17      <p>Paragraph</p>
18
19    <h2>In conclusion</h2>
20      <p>Paragraph</p>
21
22  </body>
23 </html>

```

The browser preview shows the rendered content with the following structure:

- Main title**
- Introduction
- Reasons**
 - Reason 1**
 - Paragraph
 - Reason 2**
 - Paragraph
- In conclusion**
- Paragraph

The screenshot shows a Wikipedia article titled "Hypertext Markup Language". The table of contents (Sommaire) is as follows:

- 1 Dénominations
- 2 Evolution du langage
 - 2.1 1989-1992 : Origine
 - 2.2 1993 : Apports de NCSA Mosaic
 - 2.3 1994 : Apports de Netscape Navigator
 - 2.4 1995-1996 : HTML 2.0
 - 2.5 1997 : HTML 3.2 et 4.0
 - 2.6 2000-2006 : XHTML
 - 2.7 De 2007 à nos jours : HTML 5 et abandon du XHTML 2
 - 2.8 L'avenir du HTML : sans numéro de version ?
- 3 Description de HTML
 - 3.1 Syntaxe de HTML
 - 3.2 Structure des documents HTML
 - 3.3 Éléments de HTML
 - 3.4 Attributs de HTML
 - 3.5 Jeu de caractères
 - 3.6 Les 16 techniques d'échappement
- 4 Interopérabilité de HTML
- 5 Notes et références
- 6 Voir aussi
 - 6.1 Articles connexes
 - 6.2 Liens externes

Dénominations [modifier | modifier le code]

L'[anglais](#) *Hypertext Markup Language* se traduit littéralement en langage de balisage d'hypertexte^[1]. On utilise généralement le [sigle](#) HTML, parfois même en répétant le mot « langage » comme dans « langage HTML ». Il est écrit *HyperText* pour marquer le T du sigle HTML.

Le public non averti parle parfois de HTM au lieu de HTML, HTM étant l'[extension de nom de fichier](#) tronquée à trois lettres, une limitation qu'on trouve sur d'anciens [systèmes d'exploitation](#) de Microsoft.

Évolution du langage [modifier | modifier le code]

Durant la première moitié des années 1990, avant l'apparition des technologies web comme [JavaScript](#), les [feuilles de style en cascade](#) et le [Document Object Model](#), l'évolution de HTML a dicté l'évolution du Web. HTML 4, l'évolution de HTML a fortement ralenti ; 10 ans plus tard, HTML 4 reste utilisé dans les [pages web](#). En 2008, la spécification du [HTML5](#) est à l'étude^[2], et devient d'usage courant dans la seconde moitié de l'année.

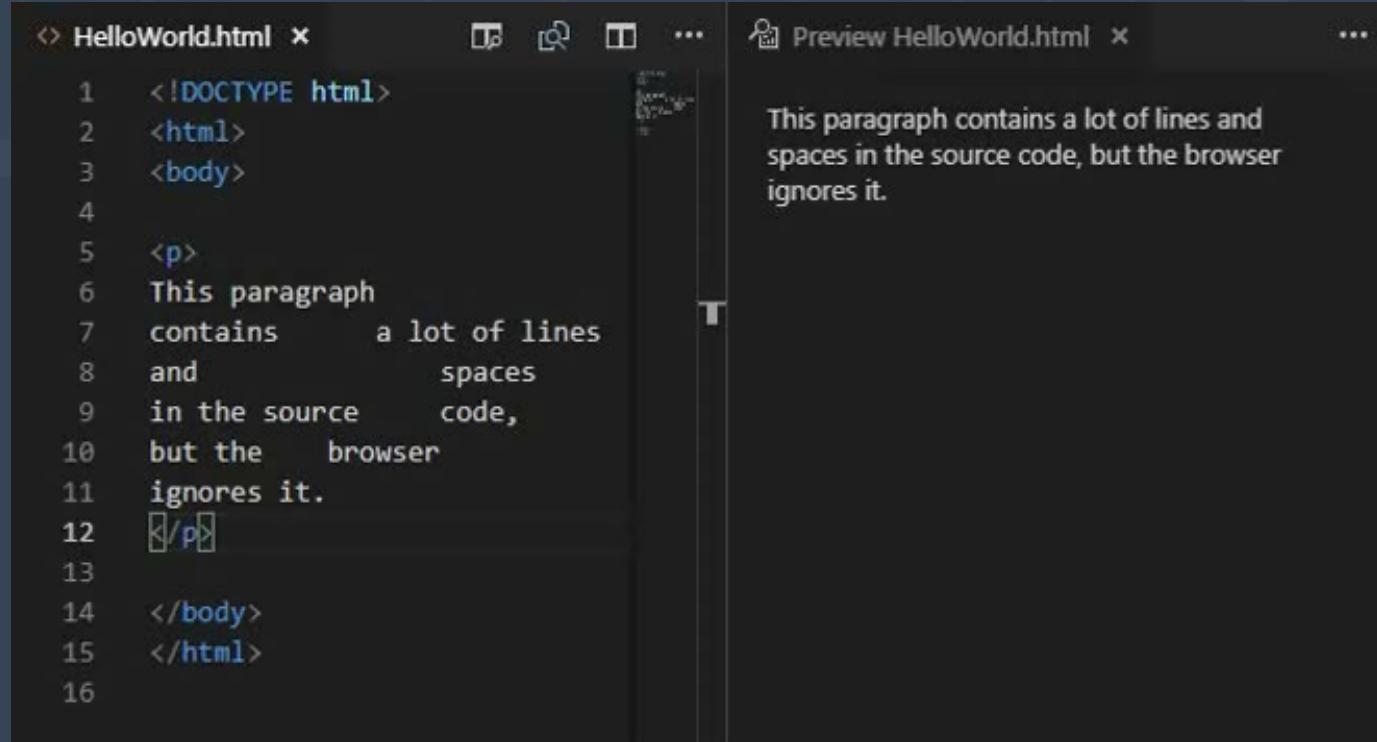
1989-1992 : Origine [modifier | modifier le code]

HTML est une des trois inventions à la base du [World Wide Web](#), avec le [Hypertext Transfer Protocol](#) (HTTP) et les [adresses web](#). HTML a été inventé pour permettre d'écrire des documents [hypertextuels](#) liant les documents entre eux. Autour d'eux, ces documents sont accès « comme web ». En août 1991, lorsque Tim Berners-Lee a annoncé publiquement le web sur l'Internet, il ne cite que le langage SGML, mais donne

Bases de l'HTML

PARAGRAPHES : BALISE <p>

- Délimite un bloc de texte
- Le navigateur ajoute des marges avant et après
- Les espaces et sauts de ligne sont ignorés
- Ne pas oublier la balise fermante !



The screenshot shows a code editor with two tabs: 'HelloWorld.html' and 'Preview HelloWorld.html'. The code in 'HelloWorld.html' is:

```
<!DOCTYPE html>
<html>
<body>
<p>
This paragraph
contains      a lot of lines
and          spaces
in the source    code,
but the        browser
ignores it.
</p>
</body>
</html>
```

The 'Preview' tab shows the resulting HTML output:

This paragraph contains a lot of lines and spaces in the source code, but the browser ignores it.

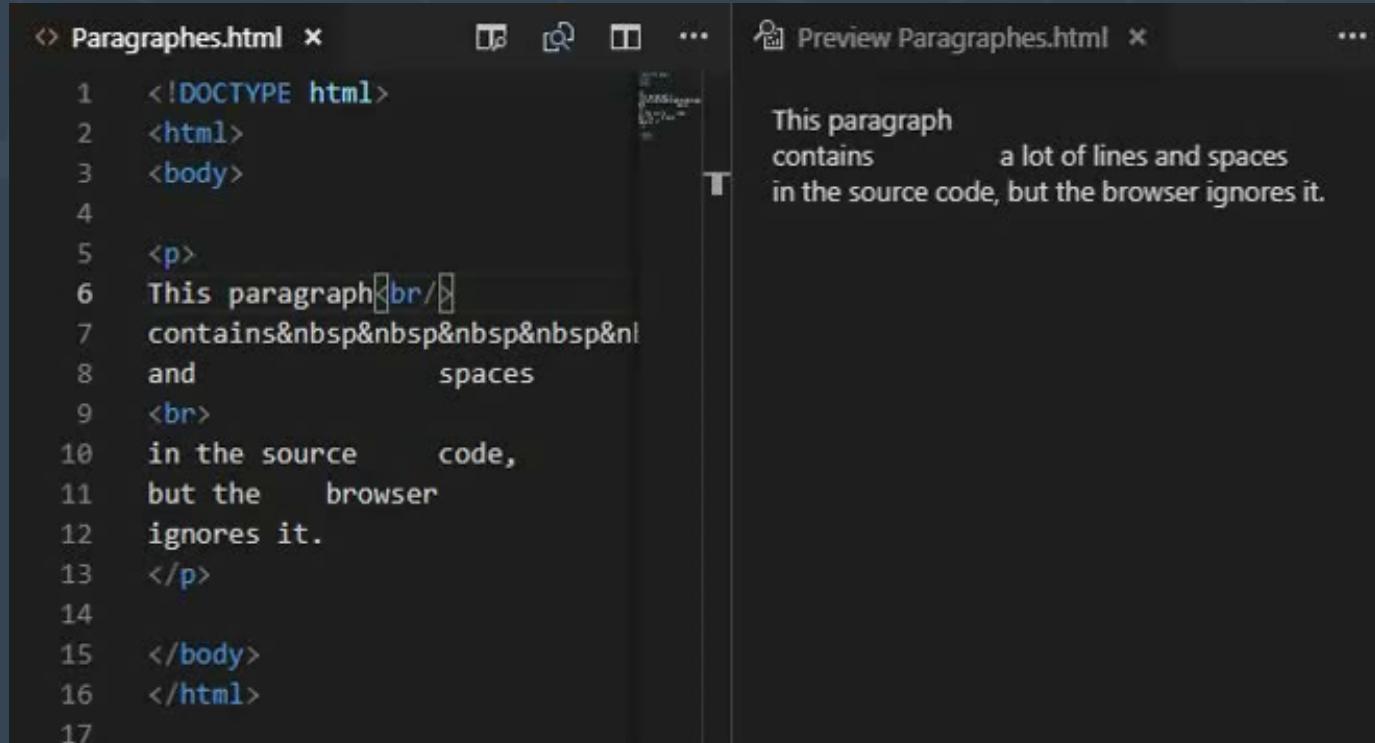
Bases de l'HTML

PARAGRAPHES : BALISE

- Forcer un saut de ligne :

- Pas pour espacer des paragraphes !!
- Forcer un espace : (non-breakable space)


 est à la fois ouvrante et fermante



The screenshot shows a code editor window titled "Paragraphes.html" and a preview window titled "Preview Paragraphes.html".

Paragraphes.html:

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4
5  <p>
6  This paragraph<br/>
7  contains&ampnbsp&ampnbsp&ampnbsp&ampnbsp&nbsps
8  and           spaces
9  <br>
10 in the source   code,
11 but the    browser
12 ignores it.
13 </p>
14
15 </body>
16 </html>
17
```

Preview Paragraphes.html:

This paragraph
contains a lot of lines and spaces
in the source code, but the browser ignores it.

Bases de l'HTML



PARAGRAPHES : BALISE <pre>

- Dans un <pre> (*preformatted*), les espaces et sauts de ligne sont préservés
- Le texte est affiché dans une **fonte à taille fixe**

The screenshot shows a code editor with two tabs: 'Paragraphes.html' and 'Preview Paragraphes.html'. The code in 'Paragraphes.html' is as follows:

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <pre>
5      My Bonnie lies over the ocean.
6
7      My Bonnie lies over the sea.
8
9      My Bonnie lies over the ocean.
10
11     Oh, bring back my Bonnie to me.
12 </pre>
13 </body>
14 </html>
15
```

The 'Preview Paragraphes.html' tab shows the rendered output:

My Bonnie lies over the ocean.
My Bonnie lies over the sea.
My Bonnie lies over the ocean.
Oh, bring back my Bonnie to me.

Formatage du texte

BALISES ****, *<i>*, ****, **

- Balises de formatage : ****, *<i>*
- Balises sémantiques : ****, **

- Différence sémantique :

<i> : Il a fini son TP *in extremis*

** : Tu as *vraiment* dit ça ?

- Penser à l'accessibilité et au contenu



<i> est aussi fréquemment utilisée pour insérer une *icône*

Le formatage du texte doit rester la responsabilité du CSS

The screenshot shows a code editor interface with a dark theme. On the left, a vertical toolbar has icons for file operations, search, and other editor functions. The main area displays a snippet of HTML code with line numbers 1 through 6. To the right of the code, the corresponding formatted text is shown in two columns: French on top and English on the bottom. The code editor's status bar at the bottom includes the number of lines (Li 6), column 1, 4 spaces, UTF-8 encoding, CRLF line endings, and HTML mode. There are also icons for a smiley face and a bell.

| | | |
|---|--|--------------------------|
| 1 | <bold>Du texte normal</bold> | Du texte normal |
| 2 | Du texte en gras | Du texte en gras |
| 3 | Du texte important | Du texte important |
| 4 | <i>Du texte en italique</i> | Du texte en italique |
| 5 | Du texte mis en évidence | Du texte mis en évidence |
| 6 | | |

Formatage du texte



AUTRES BALISES

The screenshot shows a code editor window titled "Formatage.html" and a preview window titled "Preview Formatage.html".

Code Editor Content:

```
1 <u>souligné</u> <br/>
2 <mark>surligné</mark> <br/>
3 <small>petit</small> <br/>
4 <ins>inséré</ins><br/>
5 <del>supprimé</del><br/>
6 2x<sup>2</sup>+5y<sup>4</sup><br/>
7 u<sub>1</sub> + u<sub>2</sub> + ... + u<sub>k</sub><br/>
8 <abbr title="Hypertext Markup Language">HTML</abbr>!
```

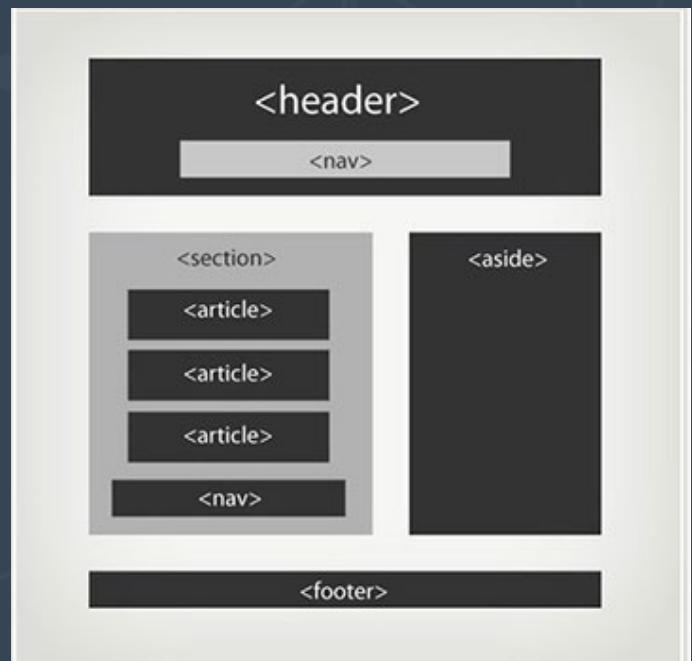
Preview Window Content:

souligné
surligné
petit
inséré
supprimé
 $2x^2+5y^4$
 $u_1 + u_2 + \dots + u_k$
HTML!
Hypertext Markup Language

Disposition d'une page web

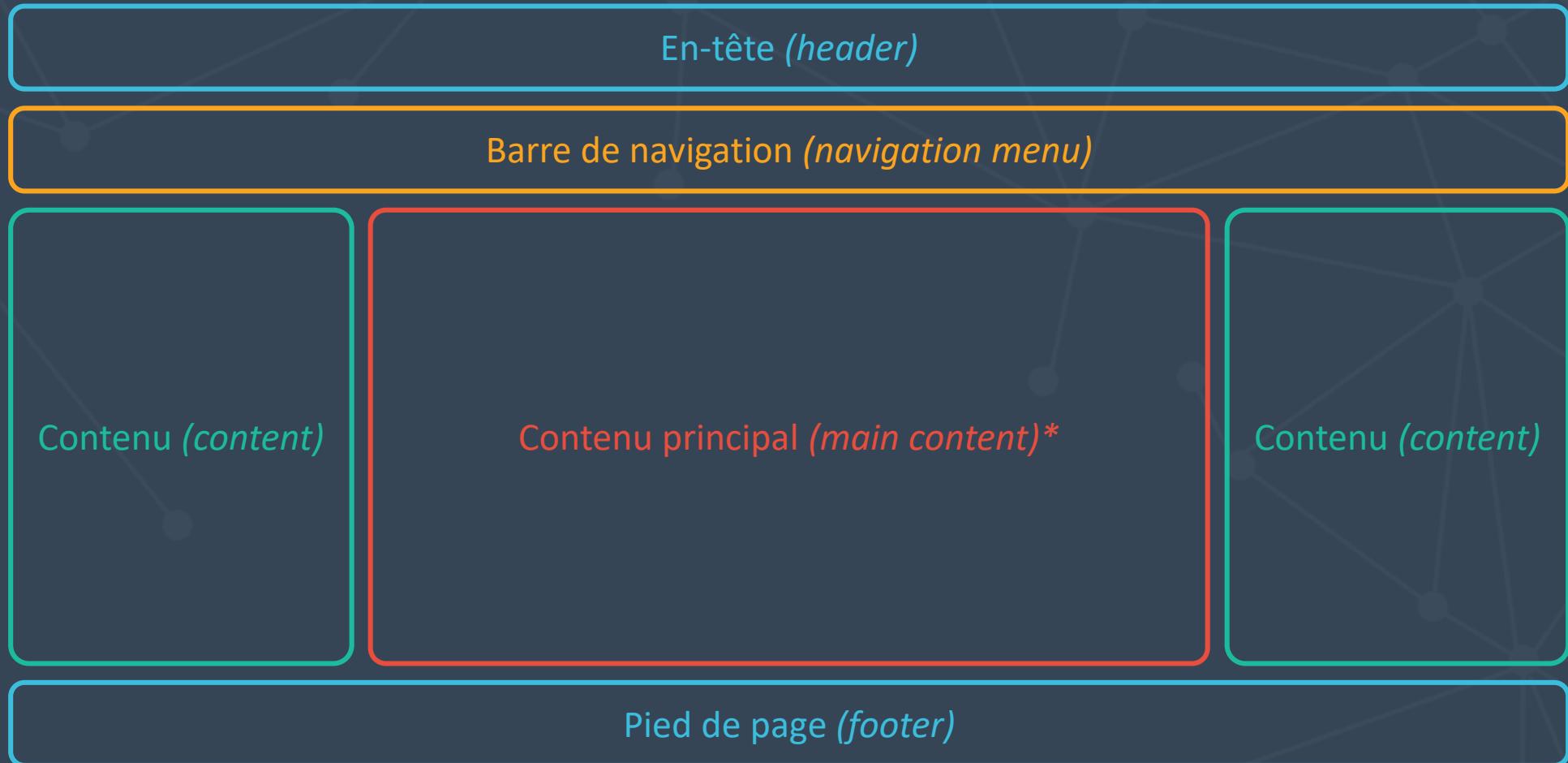
CREER DES SECTIONS

- Pour créer une section dans un document HTML, on utilise encore (trop) souvent la balise `<div>`
 - 💡 Un peu l'équivalent de la commande « *Grouper* » dans PowerPoint
- Problème : `<div>` n'a aucune *sémantique* ; sert uniquement à regrouper des contenus (par exemple pour leur affecter un style identique, ou les placer à un endroit précis sur la page)
 - ⇒ impossible de distinguer un article, un contenu annexe, une section... !
 - ⇒ HTML5 : nouvelles balises sémantiques pour répondre à ce besoin :



Disposition d'une page web : exemple

IL EN EXISTE BEAUCOUP D'AUTRES !



* On verra avec CSS comment créer du contenu multicolonnes

Disposition d'une page web

PLACE DES ELEMENTS

- Où placer `<header>` ? Avant ou dans `<body>` ?
⇒ il ne faut pas confondre `<header>` et `<head>` ! Un document HTML se compose de **deux** parties : l'en-tête (`head`) et le corps (`body`). `<header>` est une nouvelle balise sémantiques `HTML5` pour indiquer la section d'introduction d'un article, d'une section, voire de tout le document. Elle se place dans `<body> ... </body>`
- Où placer `<footer>` ? Après ou dans `<body>` ?
⇒ id.
- Où placer `<nav>` ? Dans `<header>` ou en dehors ?
⇒ « ça dépend ». Beaucoup préfèrent le placer dans `<header>`, mais ce n'est pas obligatoire. Cependant, en termes d'`accessibilité`, il est préférable de séparer les deux éléments (un lecteur d'écran fonctionnera mieux)

Disposition d'une page web

CREER DES SECTIONS : <div> ET

- Pour créer une section dans un document HTML, on utilise encore (trop) souvent la balise <div>
 - Un peu l'équivalent de la commande « *Grouper* » dans PowerPoint
-  Préférable d'utiliser les nouvelles balises sémantiques <section>, <article>, <main>, <aside>...
- La balise permet aussi de grouper des éléments, ou de cibler un élément quand il n'existe pas une meilleure balise sémantique :
 - `<p>In French, the phrase "s'il te plaît" means "please."</p>`
-  est très proche de l'élément <div>, mais
 - <div> est un élément de bloc,
 - alors que est un élément en ligne

Disposition d'une page web

ELEMENTS "BLOCK" VS ELEMENTS "INLINE"

Un élément de niveau **block** démarre au début d'une ligne et occupe tout l'espace horizontal possible :

- <div>
- <h1>...<h6>
- <p>
- <form>
- <header>
- <footer>
- <section>
- ...

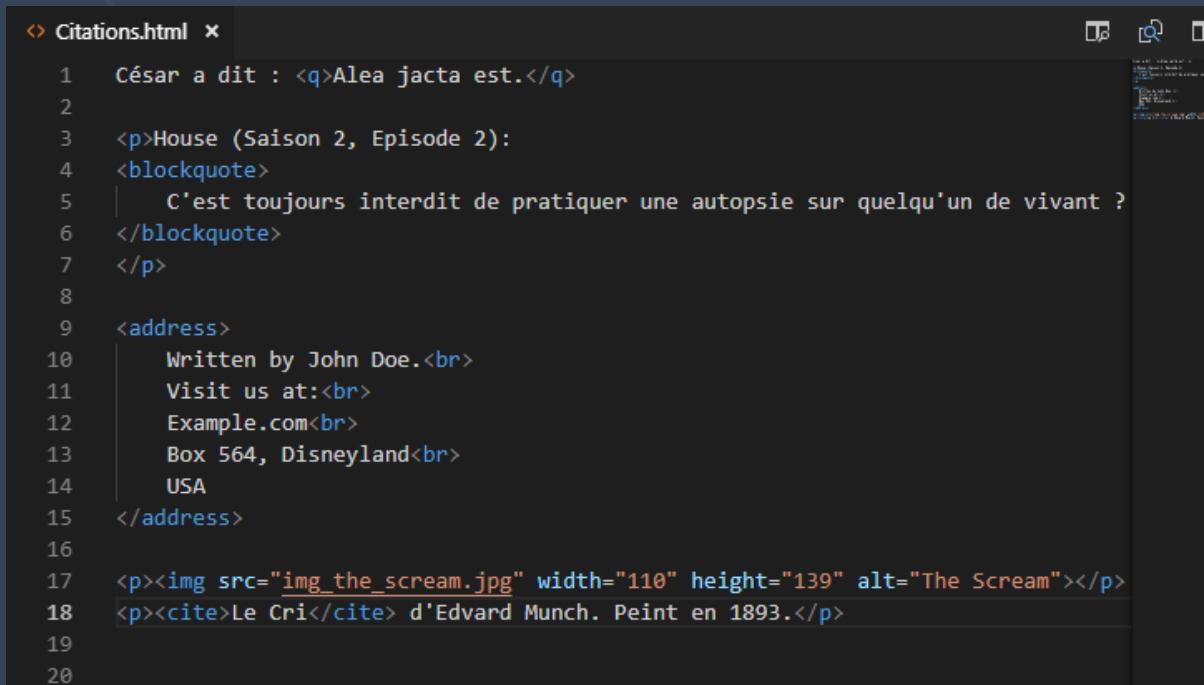
Un élément **inline** peut démarrer **n'importe où** et occupe seulement l'espace nécessaire :

-
- <a>
-
- ...

Règle d'or : on ne place jamais un élément *block* dans un élément *inline* !

Citations, adresses

- Citation courte : <q>
- Citation en évidence : <blockquote>
- Adresse : <address>
- Titre d'une œuvre ou origine d'une citation : <cite>



The screenshot shows a code editor on the left and a browser preview on the right. The code editor displays the following HTML:

```
<div>
    <h1>Citations, adresses</h1>
    <p>1 César a dit : <q>Alea jacta est.</q></p>
    <p>2</p>
    <p>3 <p>House (Saison 2, Episode 2):</p>
        <blockquote>
            <p>4 C'est toujours interdit de pratiquer une autopsie sur quelqu'un de vivant ?</p>
        </blockquote>
    <p>5</p>
    <p>6</p>
    <p>7</p>
    <p>8</p>
    <p>9 <address></p>
        <p>10 Written by John Doe.<br></p>
        <p>11 Visit us at:<br></p>
        <p>12 Example.com<br></p>
        <p>13 Box 564, Disneyland<br></p>
        <p>14 USA<br></p>
    <p>15 </address></p>
    <p>16</p>
    <p>17 <p></p></p>
    <p>18 <p><cite>Le Cri</cite> d'Edvard Munch. Peint en 1893.</p></p>
    <p>19</p>
    <p>20</p>
</div>
```

The browser preview on the right shows the rendered HTML. It includes a quote from Caesar, a blockquote from House, an address block with contact information, and a citation for "Le Cri" by Edvard Munch. A small image of the painting "The Scream" is displayed.

Insérer des images

BALISE ``

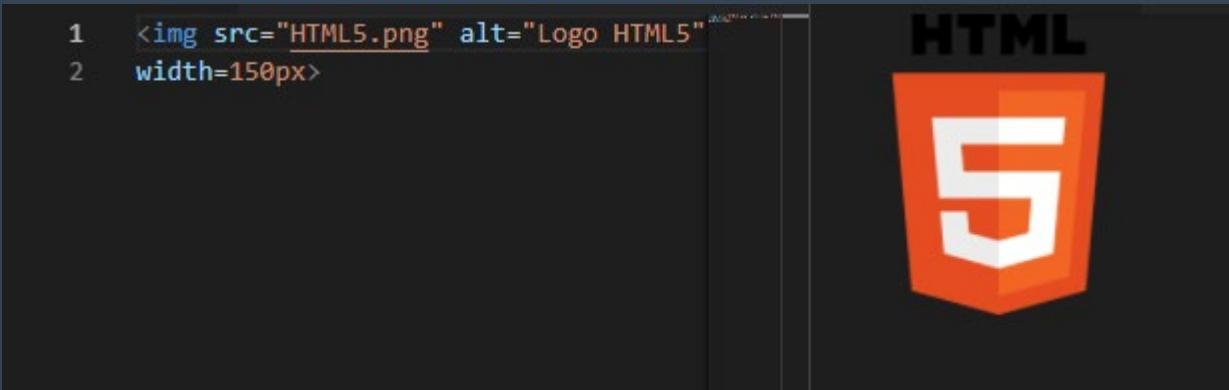
- Utilisation basique :

```

```

- Attributs principaux :

- `src` : URL (adresse) de l'image
- `alt` : texte alternatif (pour les **malvoyants**, les **moteurs de recherche** ou si problème de chargement)
- `width, height` : dimensions de l'image (on verra plus tard une meilleure méthode)



Insérer des images

UTILISATION DE L'ATTRIBUT alt

- Mauvaise utilisation de l'attribut alt :

The screenshot shows a code editor window titled "Alt.html". The code contains several tags with empty alt attributes:

```
1 
2 An anonymous user wrote:
3 <blockquote>Lorem ipsum dolor sed.</blockquote>
4 <a href="https://google.com/"></a> /
5 <a href="https://google.com/"></a>
6
```

To the right of the editor is a preview window titled "Preview Alt.html". The preview shows the visual output of the code. It includes an anonymous user's avatar (a placeholder image), the user's message ("An anonymous user wrote: Lorem ipsum dolor sed."), and two links with edit and delete icons.

- Bonne utilisation de l'attribut alt :

The screenshot shows a code editor window titled "Alt.html". The code is identical to the previous one, but the tags now have descriptive alt attributes:

```
1 
2 An anonymous user wrote:
3 <blockquote>Lorem ipsum dolor sed.</blockquote>
4 <a href="https://google.com/"></a> /
5 <a href="https://google.com/"></a>
```

To the right of the editor is a preview window titled "Preview Alt.html". The preview shows the same visual output as the previous screenshot, but the alt attributes provide meaningful descriptions for screen readers and search engines.

Ancres et hyperliens

BALISE <a>

- Utilisation basique :

```
<a href="index.html">Retourner à la page d'accueil</a>
```

- Attributs principaux :

- **href** : destination du lien
- **target** : où sera ouvert le lien
 - **_self** : dans la même fenêtre / même onglet (valeur par défaut)
 - **_blank** : dans un nouvel onglet / nouvelle fenêtre (**⚠ faille de sécurité** ; voir « noopener / noreferrer »)
- **title** : titre du document cible (affiché dans une info-bulle)
- **download** : indique que la cible doit être téléchargée et non ouverte dans le navigateur

Listes

BALISES , ,

- 3 types de listes :

| non ordonnée | ordonnée | description / définition |
|--|---|---|
| <ul style="list-style-type: none"> • Œufs • Farine • Lait <pre> Œufs Farine Lait </pre> | <ol style="list-style-type: none"> 1. Harvard 2. Stanford 3. Cambridge <pre> Harvard Stanford Cambridge </pre> | <p>Café - boisson chaude noire</p> <p>Lait - boisson froide blanche</p> <pre><dl> <dt>Café</dt> <dd>boisson chaude noire</dd> <dt>Lait</dt> <dd>boisson froid noire</dd> </dl></pre> |

- On peut changer les numéros des items d'une liste ordonnée :

```
<ol start="3">
  <li>Harvard</li>
  <li>Stanford</li>
  <li value=8>Cambridge</li>
  <li>MIT</li>
</ol>
```

3. Harvard
4. Stanford
8. Cambridge
9. MIT

- On peut changer l'aspect des puces (liste non ordonnée) ou des ordinaux (liste ordonnée) avec une propriété CSS :

```
<ul style="list-style-type:circle;">
  <li>Oeufs</li>
  <li>Farine</li>
  <li>Lait</li>
</ul>

<ol style="list-style-type:upper-roman;">
  <li>Harvard</li>
  <li>Stanford</li>
  <li>Cambridge</li>
</ol>
```

○ Oeufs
○ Farine
○ Lait
I Harvard
II Stanford
III Cambridge

- On peut imbriquer des listes, même de différents types :

```
<ol>
  <li>item 1</li>
  <li>item 2
    <ul>
      <li>sous-item 2.1</li>
      <li>sous-item 2.2</li>
    </ul>
  </li>
  <li>item 3</li>
</ol>
```

- 1. item 1
- 2. item 2
 - sous-item 2.1
 - sous-item 2.2
- 3. item 3

Barre de navigation

COMMENT CREER UN MENU SIMPLEMENT

- Avant HTML5, on utilisait souvent les listes pour créer des menus :

```
<ul>
  <li><a href="#">Accueil</a></li>
  <li><a href="#">A propos</a></li>
  <li><a href="#">Contact</a></li>
</ul>
```

- Depuis HTML5, on dispose de la balise `<nav>` pour renforcer la **sémantique** (une liste est une liste et une barre de navigation est une barre de navigation) :

```
<nav>
  <a href="#">Accueil</a>
  <a href="#">A propos</a>
  <a href="#">Contact</a>
</nav>
```

Cependant, il est courant de continuer à utiliser des listes, notamment pour un **menu hiérarchique**

Tableaux

PRÉSENTER DES DONNÉES

Un tableau simple :

```
<table>
  <caption>Tableau des âges</caption>
  <tr>
    <th>Prénom</th>
    <th>Nom</th>
    <th>Âge</th>
  </tr>
  <tr>
    <td>Rick</td>
    <td>Grimes</td>
    <td>43</td>
  </tr>
  <tr>
    <td>Negan</td>
    <td></td>
    <td>43</td>
  </tr>
</table>
```

| Tableau des âges | | |
|------------------|--------|-----|
| Prénom | Nom | Âge |
| Rick | Grimes | 43 |
| Negan | | 43 |

Tableaux



FUSIONNER DES LIGNES OU DES COLONNES

On peut fusionner des lignes ou des colonnes :

```
<table border="">
  <tr>
    <td>row 1 col 1</td>
    <td>row 1 col 2</td>
    <td>row 1 col 3</td>
  </tr>
  <tr>
    <td colspan="3">This second row spans all three columns</td>
  </tr>
  <tr>
    <td rowspan="2">This cell spans two rows</td>
    <td>row 3 col 2</td>
    <td>row 3 col 3</td>
  </tr>
  <tr>
    <td>row 4 col 2</td>
    <td>row 4 col 3</td>
  </tr>
</table>
```

| | | |
|---|-------------|-------------|
| row 1 col 1 | row 1 col 2 | row 1 col 3 |
| This second row spans all three columns | | |
| This cell spans two rows | row 3 col 2 | row 3 col 3 |
| | row 4 col 2 | row 4 col 3 |

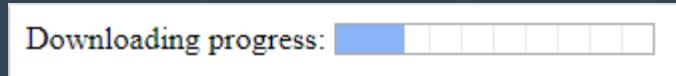
L'attribut "border" n'est là que pour mettre en évidence les différentes cellules.

On verra comment utiliser CSS pour afficher les bordures

Beaucoup d'autres balises

A EXPLORER !

- <progress> : créer une barre de progression



- <map> : créer une image avec des zones cliquables



Source : <http://formation.upyupy.fr/html-xhtml/images-map/>

- et bien d'autres !

Formulaires

UN PREMIER ELEMENT D'INTERACTION, BALISE <form>

- Pour créer un formulaire, on utilise la balise **<form>**

L'attribut **action** définit la page à contacter lorsque le formulaire est soumis (par ex. une page écrite en PHP)

- Les champs du formulaire sont créés avec la balise **<input>**

Il faut spécifier un attribut **type** ; valeurs possibles : **text, radio, submit, checkbox, email, hidden, datetime...**

L'attribut **name** est obligatoire (sinon la valeur du champ n'est pas envoyée au serveur)

```
<form action="/action_page.php" method="GET">
    Prénom :<br>
    <input type="text" name="firstname" value="Mickey"><br>
    Nom :<br>
    <input type="text" name="lastname" value="Mouse"><br><br>
    <input type="radio" name="sexe" value="homme" checked> Homme
    <input type="radio" name="sexe" value="femme" checked> Femme<br><br>
    <input type="submit" value="Submit">
</form>
```

Prénom :

Mickey

Nom :

Mouse

Homme Femme

Submit

Formulaires

QUELQUES CONTRÔLES SUPPLEMENTAIRES

Choisissez un fichier :

```
<input type="file" name="myFile"><br><br>
```

Choisissez une couleur :

```
<input type="color" name="favcolor"><br><br>
```

Quantité (entre 1 et 5) :

```
<input type="number" name="quantity" min="1" max="5"
```

Rechercher :

```
<input type="search" name="googlesearch"><br><br>
```

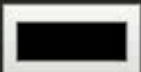
Choisissez une date :

```
<input type="week" name="week_year">
```

Choisissez un fichier :

Choisissez un fichier Aucun fichier choisi

Choisissez une couleur :



Quantité (entre 1 et 5) :

Rechercher :

Choisissez une date : Semaine --, ----

⚠ Certains contrôles ne sont pas compatibles avec tous les navigateurs !



Formulaires

QUELQUES CONTRÔLES SUPPLEMENTAIRES

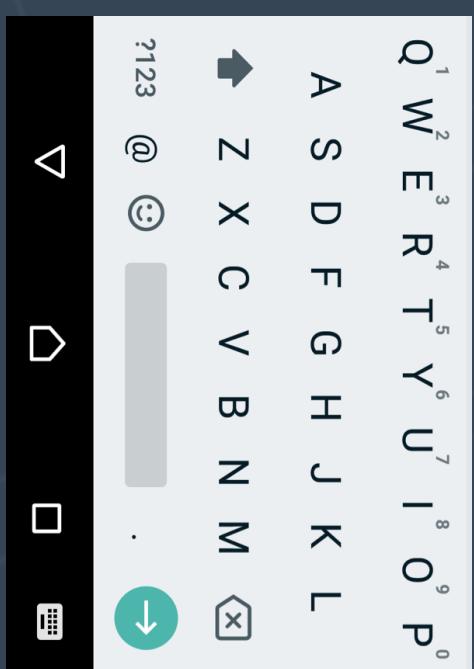
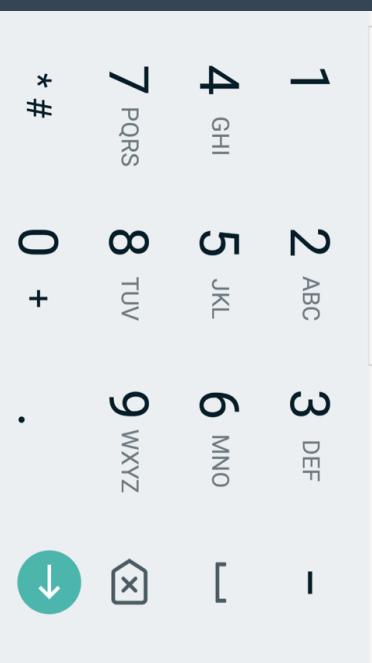
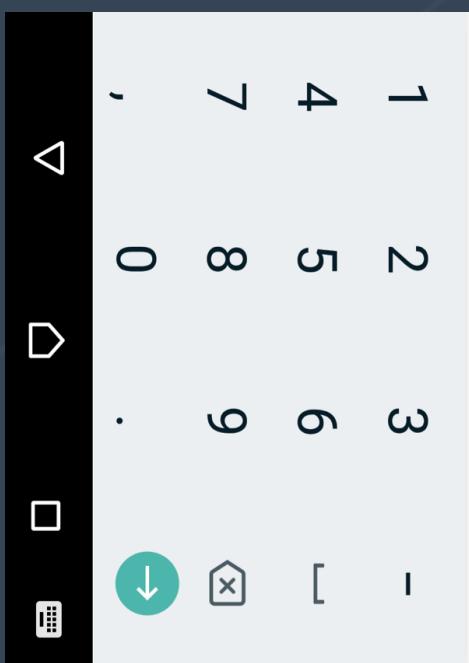
- Attention aux input de type *number* :
 - pas adaptés pour tout (ex. : codes postaux, numéros de téléphone, numéros de CB...)
 - ont un bouton d'incrémentation / décrémentation => pas toujours pertinent (et dans les anciens navigateurs, la valeur est modifiée si l'utilisateur « scrolle » au-dessus du champ de saisie !)
 - certains navigateur arondissent ou modifient le format des nombres trop grands (ex. : Chrome qui passe en notation scientifique quand on utilise le clavier)
 - en termes d'accessibilité : l'utilisateur n'a aucun retour par rapport aux caractères rejetés (lettres par ex.)
- Pour les cas précédents, on utilisera plutôt un input de type *text* en précisant un *pattern* :

```
<input type="text" id="code_postal" name="code_postal"  
pattern="[0-9]{5}" title="Code postal sur 5 chiffres">
```

Formulaires

QUELQUES CONTRÔLES SUPPLEMENTAIRES

- L'attribut *inputmode* indique aux navigateurs mobiles quel clavier afficher :



Formulaires

QUELQUES CONTRÔLES SUPPLEMENTAIRES

- L'attribut *autocomplete* spécifie si un champ de formulaire doit être autocomplété :

```
<form action="/action_page.php" autocomplete="on">  
    <label for="fname">Prénom :</label>  
    <input type="text" id="fname" name="given-name">  
</form>
```



- L'attribut *name* peut posséder l'une des valeurs suivantes :

honorable-prefix, given-name, family-name, nickname, email, username, new-password, current-password, organisation, street-address, country-name, postal-code, cc-name, cc-number, cc-exp, cc-csc, cc-type, language, tel...

Formulaires

BALISE <fieldset>

La balise <fieldset> est utilisée pour regrouper des champs de formulaire :

```
<form>
  <fieldset>
    <legend>Identité</legend>
    Prénom :<br>
    <input type="text"><br>
    Nom :<br>
    <input type="text"><br>
  </fieldset><br>
  <fieldset>
    <legend>Message</legend>
    <textarea>Votre message</textarea>
  </fieldset><br>
  <input type="submit" value="Submit">
</form>
```

The screenshot shows a web form with a dark background. It features two nested `<fieldset>` elements. The first `<fieldset>` has a legend "Identité" and contains two text input fields for "Prénom" and "Nom". The second `<fieldset>` has a legend "Message" and contains a text area labeled "Votre message". A "Submit" button is located below the second `<fieldset>`.

Identité

Prénom :

Nom :

Message

Votre message

Submit

Formulaires

LISTES DEROULANTES

- Balise `<select>` :

```
<select name="marque">
    <option value="renault">Renault</option>
    <option value="peugeot">Peugeot</option>
    <option value="citroen">Citroën</option>
    <option value="porsche">Porsche</option>
</select>
```

Renault ▾

- On peut spécifier les attributs `size` (nombre d'éléments visibles) et `multiple` (sélection multiple autorisée) :

```
<select name="marque" size="3" multiple>
    <option value="renault">Renault</option>
    <option value="peugeot">Peugeot</option>
    <option value="citroen">Citroën</option>
    <option value="porsche">Porsche</option>
</select>
```

Renault
Peugeot
Citroën

Formulaires

LES ATTRIBUTS method ET autocomplete

- La balise `<form>` dispose d'un attribut **method** ; 2 valeurs possibles : **GET** ou **POST**
 - La valeur indique sous quelle forme est envoyée la **requête HTTP** au **serveur web**
 - **GET** (valeur par défaut) :
 - pour *obtenir* des données (par exemple, un article *via* sa référence), ou pour envoyer une requête
 - les données soumises sont **visibles dans la barre d'adresse** !
 - **POST** :
 - pour *soumettre* des données
 - les données soumises ne sont **pas visibles dans la barre d'adresse** !

Commentaires

NE PAS LES NEGLIGER !

- Syntaxe :

```
<!-- commentaire -->
```

- Un commentaire peut s'étaler sur plusieurs lignes :

```
<!-- Ceci est un commentaire long...
      très long...
      vraiment très long...
-->
```

- **⚠** On ne met pas de commentaire **dans** une balise :

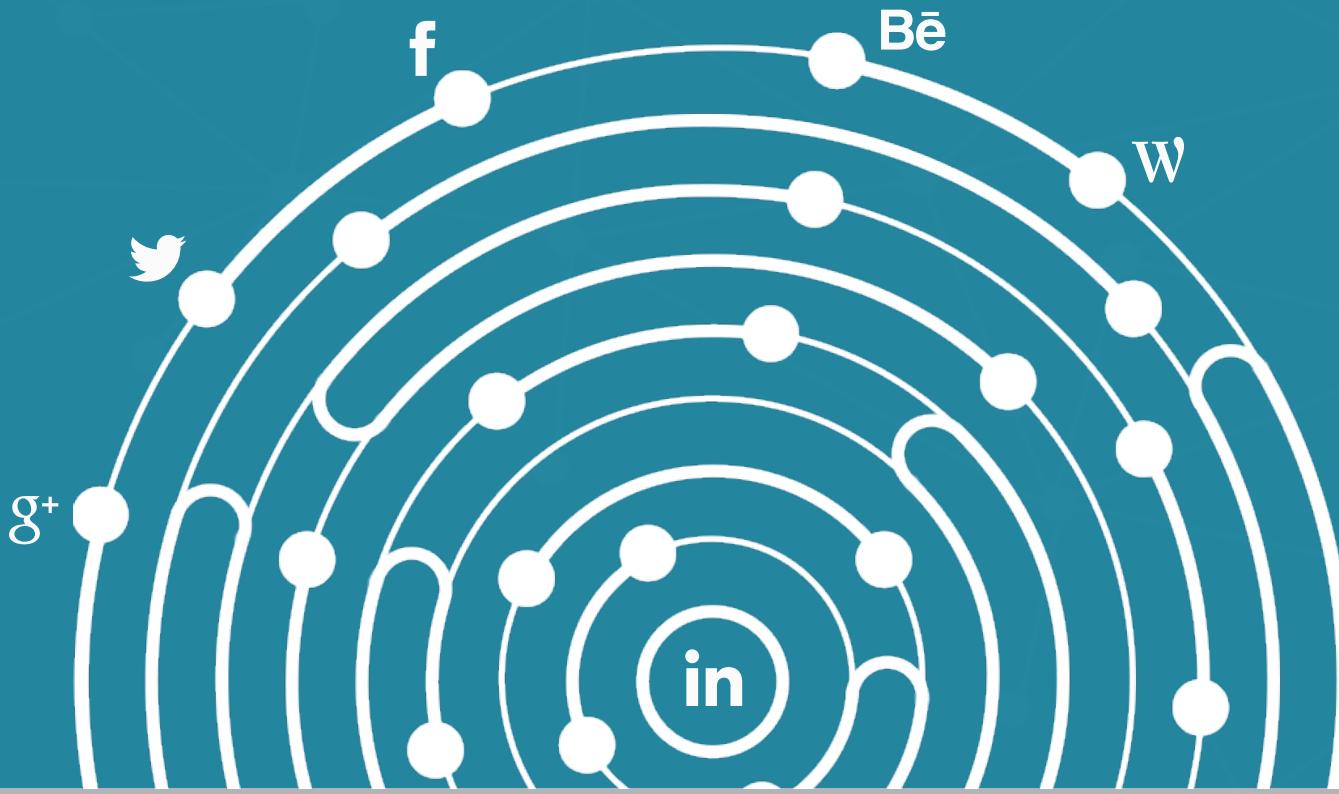
```
<h1 <!-- ça ne fonctionne pas... -->>
```

Certains outils de validations n'aiment pas non plus quand on met '--' dans un commentaire

- **Web Storage** : pour stocker des données dans le navigateur (cf. cours 4)
- **Multimédia** : dessin, sons, vidéos, YouTube, plug-ins... (cf. cours 5)
`<canvas>, <audio>, <video>, <track>...`
- **Géolocalisation** : connaître la position de l'utilisateur, et proposer des services à proximité
- **Drag & Drop** : déplacer des éléments, joindre un fichier simplement...
- **Web Workers** : effectuer des tâches en arrière-plan
- **Server-Sent Events** : recevoir des mises à jour automatiques d'un serveur

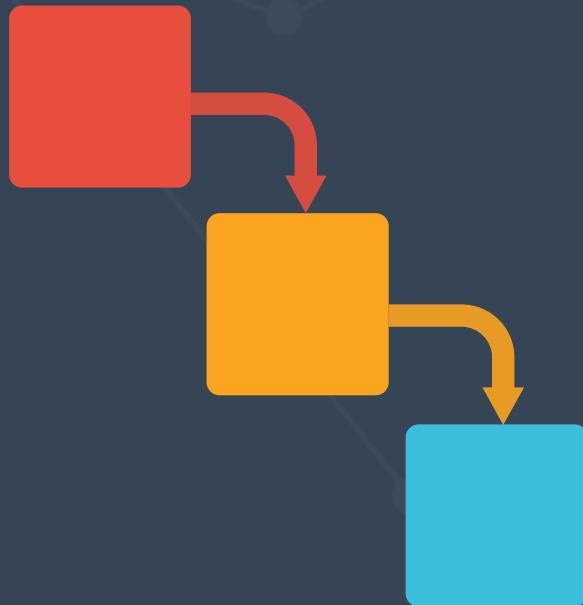


QUESTION
TIME



2^{ème} partie : CSS

CSS : Cascading Style Sheets



Les *feuilles de styles* décrivent **comment** des éléments HTML doivent être affichés en fonction du support (écran, papier, projection...).

L'utilisation de **styles** permet de modifier très simplement l'apparence d'un site web (ex. : https://www.w3schools.com/css/css_intro.asp)

3 manières d'ajouter du CSS :

- en *ligne* :

```
<h1 style="color:blue;">This is a Blue Heading</h1>
```

- en *interne*, dans la section *head* du document HTML :

```
<head><style>h1 {color: blue;}</style></head>
```

- en *externe*, dans un fichier **séparé** (à privilégier !!!) :

```
<head><link rel="stylesheet" href="styles.css"></head>
```



CSS : Cascading Style Sheets

Il est possible de charger plusieurs feuilles de style, ou de spécifier plusieurs styles pour un élément :

```
<head>
  <style>h1 {color: red;}</style>
</head>
<body style="color: blue">
  <h1>Titre</h1>
```

Dans ce cas, c'est le **dernier** style d'un élément donné qui s'applique et qui remplace les déclarations précédentes

De plus, les éléments *héritent* des styles de leur parent !

La *cascade de styles*, c'est le fait qu'une règle de priorité n remplace une règle de priorité $n-1$, qui remplace une règle de priorité $n-2$, etc.

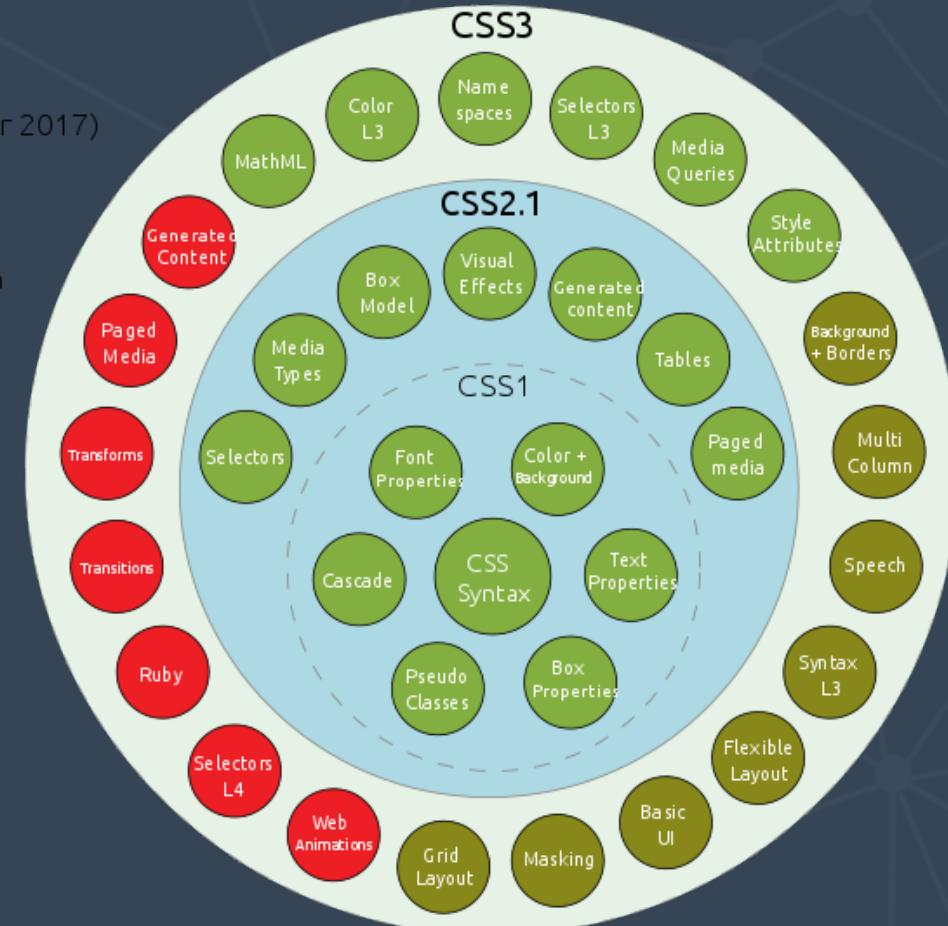
style en ligne > styles interne et externe > style par défaut du navigateur

CSS évolue lentement !

CSS3

Taxonomy & Status (September 2017)

-  W3C Recommendation
 -  Candidate Recommendation
 -  Last Call
 -  Working Draft
 -  Obsolete or inactive



Source : Wikipédia

CSS : Cascading Style Sheets

UN PREMIER EXEMPLE

The screenshot shows a code editor interface with three tabs: 'HelloCSS.html', '# styles.css', and 'Preview HelloCSS.html'.

HelloCSS.html:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 | <link rel="stylesheet" href="styles.css">
5 </head>
6 <body>
7
8 <h1>Ceci est un en-tête</h1>
9 <p>Et ceci est un paragraphe.</p>
10
11 </body>
12 </html>
```

styles.css:

```
1 body {
2     background-color: powderblue;
3 }
4 h1 {
5     color: blue;
6 }
7 p {
8     color: red;
9 }
```

Preview HelloCSS.html:

Ceci est un en-tête
Et ceci est un paragraphe.

CSS : Cascading Style Sheets

UN AUTRE EXEMPLE

The screenshot shows a code editor with three tabs: `HelloCSS.html`, `styles.css`, and `Preview HelloCSS.html`.

`HelloCSS.html` content:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link rel="stylesheet" href="styles.css">
5 </head>
6
7 <body>
8   <h1>Ceci est un en-tête</h1>
9   <p>Et ceci est un paragraphe.</p>
10 </body>
11 </html>
```

`styles.css` content:

```
1 h1 {
2   color: #15ff00;
3   font-family: verdana;
4   font-size: 80%;
5 }
6 p {
7   color: red;
8   font-family: courier;
9   font-size: 160%;
10}
```

`Preview HelloCSS.html` content:

Ceci est un en-tête
Et ceci est un paragraphe.

CSS : Cascading Style Sheets

SYNTAXE

Une règle CSS a toujours la même syntaxe :



Attention : pas d'espace entre une valeur suivie d'une unité de mesure !

 Un fichier CSS peut contenir des commentaires ; syntaxe identique à C, Java... :

```
/* Ceci est un commentaire */
```

CSS : Cascading Style Sheets

ENCORE UN AUTRE EXEMPLE

On peut facilement changer le style des paragraphes (bordure, arrière-plan...):

The screenshot shows a code editor with three tabs:

- HelloCSS.html**: Contains the following HTML code:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <link rel="stylesheet" href="styles.css">
5   </head>
6
7   <body>
8     <h1>Ceci est un en-tête</h1>
9     <p>Et ceci est un paragraphe.</p>
10    <p>Et ceci est un paragraphe.</p>
11    <p>Et ceci est un paragraphe.</p>
12  </body>
13 </html>
```
- # styles.css**: Contains the following CSS code:

```
1 p {
2   border: 1px solid powderblue;
3   padding: 20px;
4   background-color: purple
5 }
```
- Preview HelloCSS.html**: Shows the rendered output:

Ceci est un en-tête

Et ceci est un paragraphe.

Et ceci est un paragraphe.

Et ceci est un paragraphe.

Mais comment distinguer différents **types** de paragraphes ?

Notion de classe et d'id

💡 Appliquer un style pour une **classe** particulière d'éléments : **class**

The screenshot shows a code editor with three tabs: 'HelloCSS.html', 'styles.css', and 'Preview HelloCSS.html'. The 'HelloCSS.html' tab contains the following HTML code:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link rel="stylesheet" href="styles.css">
5 </head>
6
7 <body>
8   <p>Ceci est un paragraphe.</p>
9   <p>Ceci est un paragraphe.</p>
10  <p class="error">Ceci est une erreur.</p>
11  <p>Ceci est un paragraphe.</p>
12  <p class="error">Ceci est une erreur.</p>
13 </body>
14
15 </html>
```

The 'styles.css' tab contains the following CSS code:

```
1 p.error {
2   color: red;
3 }
```

The 'Preview HelloCSS.html' tab shows the rendered output:

Ceci est un paragraphe.
Ceci est un paragraphe.
Ceci est une erreur.
Ceci est un paragraphe.
Ceci est une erreur.

💡 Appliquer un style pour **un** élément particulier : **id** (l'id doit être unique !)

| | |
|---|--|
| <pre><p id="p01"> Je suis unique </p></pre> | <pre>#p01 { color: blue; }</pre> |
|---|--|

CSS : Cascading Style Sheets

SYNTHESE DES DIFFERENTS SELECTEURS

Les différents *sélecteurs CSS* :

| Type | Exemple | La règle s'applique à |
|---------------------|---------|--|
| élément | p | tous les paragraphes de la page |
| id | #first | l'élément (unique) qui a pour id first |
| classe | .intro | tous les éléments de la classe intro |
| éléments de classe | p.intro | seulement les <i>paragraphes</i> de la classe intro |
| sélecteur universel | * | Tous les éléments HTML |
| groupe d'éléments | div, p | toutes les balises <div> et <p> |

🔗 https://www.w3schools.com/css/css_selectors.asp

🔗 <https://kittygiraudel.github.io/selectors-explained/>

CSS : Cascading Style Sheets

SYNTHESE DES DIFFERENTS COMBINEATEURS

Les différents *combinateurs CSS* :

| Type | Exemple | La règle s'applique à |
|--------------------------------------|--|---|
| Descendant | <code>div p</code> | seulement les <code><p></code> qui sont à <i>l'intérieur*</i> d'un <code><div></code> |
| enfant | <code>div > p</code> | seulement les <code><p></code> dont le père est un <code><div></code> |
| frère cadet | <code>div + p</code> | seulement les <code><p></code> situés immédiatement après un <code></div></code> |
| petit frère | <code>div ~ p</code> | tous les <code><p></code> venant après un <code></div></code> |
| état particulier (pseudo-classes) | <code>p:hover</code> <code>input:focus</code> | tout paragraphe survolé par la souris tout champ de formulaire actif |
| pseudo-élément | <code>p::after</code> | contenu inséré après le contenu de chaque <code><p></code> |

* À l'intérieur est à prendre au sens large : la balise `<p>` peut être un *enfant* de la balise `<div>`, mais aussi un petit-enfant, un arrière-petit-enfant, etc.

🔗 voir https://www.w3schools.com/cssref/css_selectors.asp pour de nombreux exemples

CSS : Cascading Style Sheets

SYNTHESE DES DIFFERENTS SELECTEURS D'ATTRIBUTS

| Type | Exemple | La règle s'applique à |
|--|-------------------|--|
| attribut | a[target] | toute balise <code><a></code> qui possède l'attribut target |
| attribut valué | a[target=value] | La valeur de l'attribut target vaut value |
| attribut contenant... | a[target*=value] | la valeur de l'attribut target contient value |
| attribut commençant par... | a[target^=value] | la valeur de l'attribut target commence par value |
| attribut se terminant par... | a[target\$=value] | la valeur de l'attribut target finit par value |
| attribut contenant un mot donné | a[target~=value] | la valeur de l'attribut target contient le mot value |

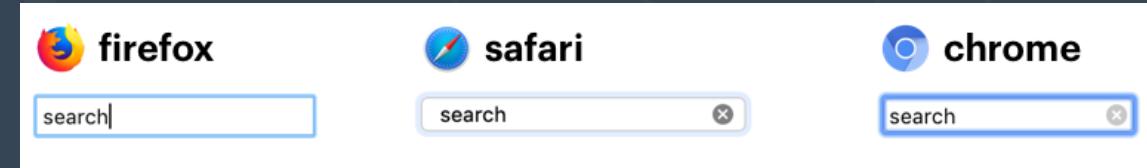


voir https://www.w3schools.com/cssref/css_selectors.asp pour de nombreux exemples

Reset CSS / CSS Normalizer

COMMENT OBTENIR LE MÊME STYLE SUR TOUS LES NAVIGATEURS ?

💡 Les navigateurs ont des styles par défaut, différents les uns des autres :



Méthode brutale : Reset CSS (déconseillée) :

```
* {  
    margin: 0;  
    padding: 0;  
    border: 0;  
    font-family: sans-serif;  
    ...  
}
```

Meilleure solution : utiliser une feuille de style normalisée : (ex. : necolas.github.io/normalize.css/)

Héritage des styles

💡 CSS est basé sur le modèle *Parent-enfants* : chaque élément enfant reçoit en héritage les styles de son élément parent

Exemples : <html> est parent de <body>, <table> est parent de <tr>, lui-même parent de <td>

⚠ Certains styles ne s'héritent pas : marges, padding...

💡 On peut appliquer l'héritage aux classes et id (attention à l'ordre) :

```
.menu li {  
    propriétés  
}
```

éléments contenus
dans la classe *menu*

```
li .menu {  
    propriétés  
}
```

éléments de classe *menu*
contenus dans un

Couleurs

https://www.w3schools.com/css/css_colors.asp

Une couleur peut être spécifiée par son nom (140 noms standard) :

| | | | |
|--------|-----------|------------|----------------|
| Tomato | Orange | DodgerBlue | MediumSeaGreen |
| Gray | SlateBlue | Violet | LightGray |

Ou par un code couleur au format :

- **RGB** : 3 composantes entre 0 et 255 pour le Rouge, le Vert et le Bleu (ex. : `rgb(185, 57, 126)`)
- HEX : id. mais les valeurs sont en **hexadécimal**, i.e. entre 00 et FF (ex. : `#b9397E`)
 $(185_{10} = 11 \times 16 + 9 = B9_{16}) \quad 7E_{16} = 7 \times 16 + 14 = 112 + 14 = 126_{10})$
- **RGBA** : id. RGB + une composante pour la transparence (entre 0 et 1) (ex. : `rgba(185, 57, 126, 0.5)`)
- **HSL** : 3 composantes entre 0 et 100 pour le %age de teinte, saturation et luminosité (ex. : `hsl(67, 35, 18)`)

On peut créer facilement des gradients de couleurs

Contrôler les dimensions

https://www.w3schools.com/css/css_units.asp

On peut facilement contrôler les dimensions de chaque élément, de manière **absolue** ou **relative** :

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <link rel="stylesheet" href="styles.css">
5  </head>
6  <body>
7  |   <p>Cet élément div a une hauteur de
8  |       200px et une largeur de 50% :
9  |   </p>
10 |   <div></div>
11 </body>
12 </html>
13
14
```

```
1  div {
2  |   height: 200px;
3  |   width: 50%;
4  |   background-color: #ccc;
5  }
6
7
8
9
10
11
12
13
14
```

Cet élément div a une hauteur de 200px et une largeur de 50% :



Mesures des longueurs

- absolues : **cm**, **mm**, **in**, **pt** (=1/72 in), **pc** (=12 pt), **px** (dépend du matériel)
- relatives : **em** (taille de police courante de l'élément), **%** (par rapport à l'élément **parent**) , **fr** (fraction de l'espace total), **vw** / **vh** (% de largeur / hauteur de la fenêtre)... ([exemple](#))



Contrôler les dimensions

https://www.w3schools.com/css/css_units.asp

Problème de l'unité **em** : la taille de la police d'un élément est **relative à celle de son parent**

⇒ potentiellement un effet de **cascade** qui donne des résultats indésirables / inattendus

Exemples :

- règle **ul { font-size: 0.75em; }** sur des listes imbriquées
- règle **div { font-size: 1.5em; }** sur des *div* imbriqués

Solution possible :

ajouter une règle supplémentaire du type **ul ul { font-size: 1em; }** ⇒ contraignant

Meilleure solution :

utiliser l'unité **rem** (pour *root em*) (utilisé dans les frameworks type *Bootstrap, Material Design, etc.*)



Contrôler les dimensions

https://www.w3schools.com/css/css_units.asp

Le *root element*, c'est la balise `<html>` ; le *rem*, c'est la taille relative par rapport au root element

Exemple : $1.2 \text{ rem} = 1,2 \times \text{la taille de la police du root element}$

💡 Avec *rem*, toutes les tailles font référence à la taille de l'élément racine
⇒ plus besoin de gérer les cas d'éléments imbriqués dans des règles séparées

❓ Est-il possible de spécifier une font-size en *rem* pour l'élément `<html>` ? Si oui, à quoi fait-elle référence ?

💡 Astuce de Snook pour convertir des *rem* en *px* : suppose que la taille de police par défaut dans les navigateurs est 16px
⇒ on la ramène à 62,5 % soit 10px
⇒ Ainsi, $1\text{rem} = 10\text{px}$, $1.2\text{rem} = 12\text{px}$, $2.7\text{rem} = 27\text{px}$, etc.

Arrière-plans

https://www.w3schools.com/css/css_background.asp

5 propriétés permettent de définir l'arrière-plan :

- **background-color**
- **background-image**
- **background-repeat** : répète l'image horizontalement, verticalement, ou dans les deux sens
- **background-attachment** : l'arrière-plan est fixe quand on fait défiler la page
- **background-position**

💡 Il existe un raccourci : la propriété **background**, à condition de respecter l'ordre ci-dessus :

```
body {  
    background: #ffffff url("img_tree.png") no-repeat right top;  
}
```

Rem. : on n'est pas obligé de spécifier toutes les propriétés, du moment que l'ordre est respecté

Bordures

https://www.w3schools.com/css/css_border.asp

Il existe de nombreuses propriétés sur les bordures, qui permettent de configurer :

- le style : **border-style**
- l'épaisseur : **border-width**
- la couleur : **border-color**
- l'arrondi : **border-radius**
- les côtés : **border-left**, **border-top**...

`border-color: cyan`

`border-bottom: dashed tomato 5px`

`border: 3px solid orange; border-radius: 10px;`



Marges et remplissages

https://www.w3schools.com/css/css_margin.asp et https://www.w3schools.com/css/css_padding.asp

Les marges permettent de définir un espace autour d'un élément, **à l'extérieur** des bordures :

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <link rel="stylesheet" href="styles.css">
5  </head>
6  <body>
7  |   <div>Cet élément a une marge de 70px.</div>
8  </body>
9  </html>
10 .
```

```
1  div {
2  |   margin: 70px;
3  |   border: 1px solid #4CAF50;
4  }
```

Cet élément a une marge de 70px.

Le remplissage ou *padding* permet de définir un espace autour d'un élément, **à l'intérieur** des bordures :

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <link rel="stylesheet" href="styles.css">
5  </head>
6  <body>
7  |   <div>Cet élément a un padding de 70px.</div>
8  </body>
9  </html>
10 .
```

```
1  div {
2  |   padding: 70px;
3  |   border: 1px solid #4CAF50;
4  }
```

Cet élément a un padding de 70px.

Marges et remplissages

https://www.w3schools.com/css/css_margin.asp et https://www.w3schools.com/css/css_padding.asp

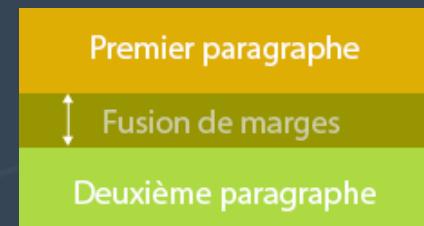
Comme pour les bordures, on peut spécifier la marge / le remplissage pour chacun des 4 côtés d'un élément :

- margin-top / padding-top
- margin-right / padding-right
- margin-bottom / padding-bottom
- margin-left / padding-left

La valeur **margin : auto** permet de centrer (horizontalement) automatiquement un élément ([exemple](#))

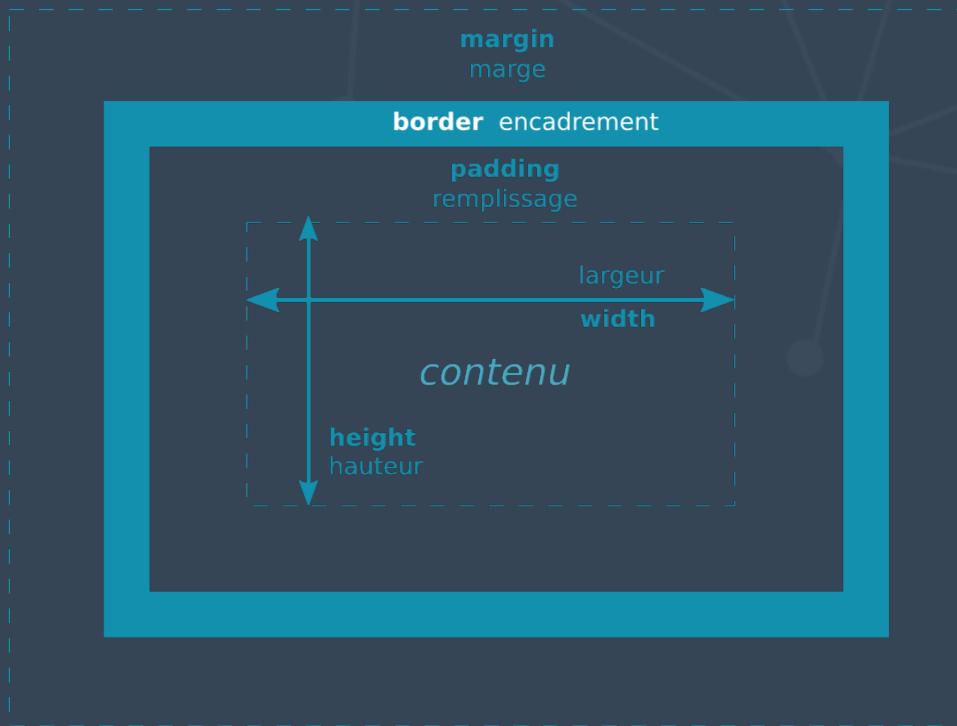
La valeur **inherit** permet d'hériter de la valeur d'une propriété depuis l'élément parent ([exemple](#))

 Dans certains cas (voir [ici](#)) les marges hautes et basses sont **fusionnées**



Modèle en boîtes

Tous les éléments HTML peuvent être vus comme des « boîtes » :



width et **height** sont les dimensions du **contenu** !

Largeur *réelle* d'une boîte = **width** + **paddings** + **borders** + **margins**

Hauteur *réelle* d'une boîte = **height** + **paddings** + **borders** + **margins**

Modèle en boîtes

Ceci conduit parfois à des effet indésirables :

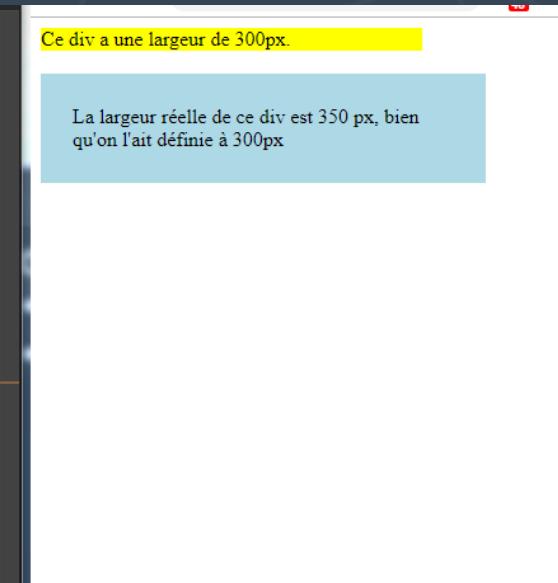
```
<!DOCTYPE html>
<html>

<head>
    <link rel="stylesheet" href="css1.css">
</head>

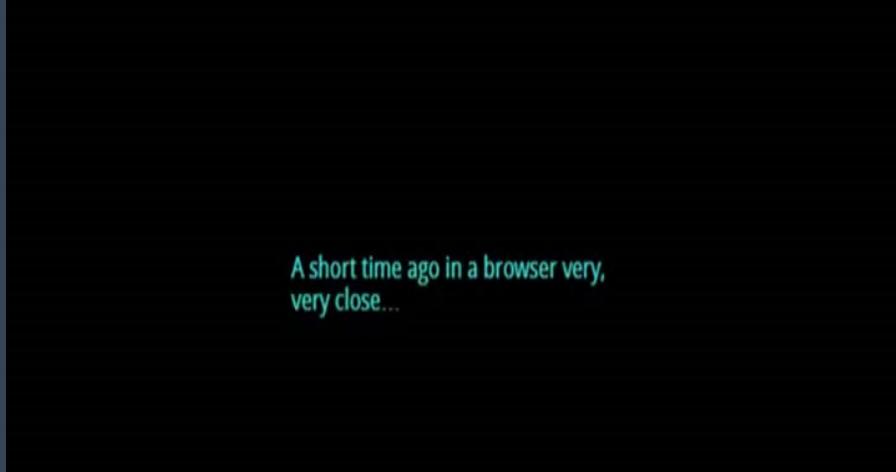
<body>
    <div class="ex1">
        Ce div a une largeur de 300px.
    </div>
    <br>
    <div class="ex2">
        La largeur réelle de ce div est 350 px,
        bien qu'on l'ait définie à 300px
    </div>
</body>

</html>
```

```
1  div.ex1 {
2      width: 300px;
3      background-color: yellow;
4  }
5
6  div.ex2 {
7      width: 300px;
8      padding: 25px;
9      background-color: lightblue;
10 }
11
12
13
14
15
16
17
18
19
20
```



 Pour préserver la largeur indiquée, on utilise la propriété **box-sizing** avec la valeur **border-box** ([exemple](#))



<https://codepen.io/thatbram/pen/KuHsl>
<https://www.sitepoint.com/css3-starwars-scrolling-text/>

Tous les éléments relatifs au texte sont paramétrables :

- fonte (**font**)
- couleur (**color**)
- espacement (**letter-spacing**, **word-spacing**)
- alignement (**text-align**)
- indentation (**text-indent**)
- transformation (**text-transform**)
- hauteur de ligne (**line-height**)
- ombrage (**text-shadow**)
- ...

Les liens peuvent facilement être transformés en boutons plus esthétiques :

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <link rel="stylesheet" href="styles.css">
5  </head>
6  <body>
7  |   <a href="index.html" target="_blank">
8  |       Ceci est un lien
9  |   </a>
10 </body>
11 </html>
```

```
1  a:link, a:visited {
2      background-color: #f44336;
3      color: white;
4      padding: 14px 25px;
5      text-align: center;
6      text-decoration: none;
7      display: inline-block;
8  }
9
10
11 a:hover, a:active {
12     background-color: red;
13 }
```

Ceci est un lien

Les propriétés CSS permettent de :

- personnaliser les marqueurs : **list-style-type** (`circle`, `square`, `upper-roman`, `lower-alpha...`)
- utiliser une image à la place d'une puce : **list-style-image** (`url`)
- ajouter une couleur d'arrière-plan (aux items ou à la liste entière) : **background** (`couleur`)
- spécifier l'emplacement des marqueurs : **list-style-position** (`inside`, `outside`)

The screenshot shows a code editor interface with three tabs:

- CSS_list.html**: Contains the following HTML code:

```
<link rel="stylesheet" href="styles.css">
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ul>
```
- styles.css**: Contains the following CSS code:

```
ul {
  list-style-image: url('images/sqpurple.png');
  background: #rgb(53, 69, 86);
  padding: 10px;
  list-style-position: outside;
}

ul li {
  background: #indigo;
  padding: 10px;
  margin: 10px;
}
```
- Preview CSS_list.html**: Shows the rendered output of the HTML and CSS. It displays a purple list with three items: "Coffee", "Tea", and "Coca Cola". Each list item has a purple square icon to its left and a purple background.

Balise HTML **<table>** :

- insère un élément considéré **sémantiquement** comme un tableau (son **contenu**)
- n'a **pas** pour rôle de gérer **comment** sera affiché le tableau (sa **mise en forme**)

En particulier, la balise **<table>** n'affiche aucune bordure (ce n'est **pas** son job !) :



The screenshot shows a table with three columns: Prénom, Nom, and Âge. The table has two rows of data: one for Rick Grimes (43 years old) and one for Negan (43 years old). The table is displayed without any visible borders.

| Prénom | Nom | Âge |
|--------|--------|-----|
| Rick | Grimes | 43 |
| Negan | | 43 |

```
<table>
  <caption>Tableau des âges</caption>
  <tr>
    <th>Prénom</th>
    <th>Nom</th>
    <th>Âge</th>
  </tr>
  <tr>
    <td>Rick</td>
    <td>Grimes</td>
    <td>43</td>
  </tr>
  <tr>
    <td>Negan</td>
    <td></td>
    <td>43</td>
  </tr>
</table>
```

Tableaux

BORDURES

Avec CSS, les bordures peuvent être ajoutées séparément aux balises `<table>`, `<th>` et `<td>` :

The screenshot shows a code editor with three tabs:

- `CSS_table.html`: Contains the HTML structure of a table with two rows and four columns, each containing two `td` elements.
- `styles.css`: Contains the CSS rules for styling the table, th, and td elements.
- `Preview CSS_table.html`: Shows the rendered table with different border styles applied to the table, th, and td elements.

`CSS_table.html` content:

```
3 <table>
4   <caption>Tableau des âges</caption>
5   <tr>
6     <th>Prénom</th>
7     <th>Nom</th>
8     <th>Age</th>
9   </tr>
10  <tr>
11    <td>Rick</td>
12    <td>Grimes</td>
13    <td>45</td>
14  </tr>
15  <tr>
16    <td>Negan</td>
17    <td></td>
18    <td>50</td>
19  </tr>
20 </table>
```

`styles.css` content:

```
1 table {
2   border: 1px ridge tomato;
3   font-size: 24px;
4 }
5 th {
6   border: 1px dashed chartreuse;
7 }
8 td {
9   border: 1px solid;
10}
```

`Preview CSS_table.html` content:

| Prénom | Nom | Age |
|--------|--------|-----|
| Rick | Grimes | 45 |
| Negan | | 50 |

La propriété `border-collapse: collapse` permet de fusionner les bordures :

The screenshot shows a code editor with three tabs:

- `CSS_table.html`: Contains the HTML structure of a table with two rows and four columns, each containing two `td` elements.
- `styles.css`: Contains the CSS rules for styling the table, th, and td elements, including `border-collapse: collapse`.
- `Preview CSS_table.html`: Shows the rendered table with the borders collapsed, appearing as a single thin border around the entire table cell.

`CSS_table.html` content:

```
3 <table>
4   <caption>Tableau des âges</caption>
5   <tr>
6     <th>Prénom</th>
7     <th>Nom</th>
8     <th>Age</th>
9   </tr>
10  <tr>
11    <td>Rick</td>
12    <td>Grimes</td>
13    <td>45</td>
14  </tr>
15  <tr>
16    <td>Negan</td>
17    <td></td>
18    <td>50</td>
19  </tr>
20 </table>
```

`styles.css` content:

```
1 table {
2   border: 1px ridge tomato;
3   font-size: 24px;
4   border-collapse: collapse;
5 }
6 th {
7   border: 1px dashed chartreuse;
8 }
9 td {
10  border: 1px solid;
11}
```

`Preview CSS_table.html` content:

| Prénom | Nom | Age |
|--------|--------|-----|
| Rick | Grimes | 45 |
| Negan | | 50 |

Tableaux

TABLEAUX ZÉBRES

Pour « zébrer » un tableau, on utilise le sélecteur `tr:nth-child(valeur)` où `valeur` vaut :

- `even`
- `odd`
- une formule du type `a n + b`

The image shows a code editor window on the left and a browser preview window on the right.

Code Editor (styles.css):

```
# styles.css x
1  table {
2    border: 1px solid;
3    font-size: 24px;
4    border-collapse: collapse;
5  }
6
7  th {
8    width: 150px;
9  }
10
11 tr:nth-child(3n+1) {
12   background-color: gray;
13 }
```

Browser Preview:

Tableau des âges

| Prénom | Nom | Âge |
|----------|--------|-----|
| Rick | Grimes | 45 |
| Negan | | 50 |
| Glenn | Rhee | 28 |
| Maggie | Greene | 26 |
| Michonne | | 37 |
| Carl | Grimes | 15 |

Tableaux

MODIFIER L'ALIGNEMENT D'UNE COLONNE

On utilise le sélecteur `tr td:nth-of-type(valeur)` où `valeur` vaut :

- `even`
- `odd`
- une formule du type `a n + b`

The screenshot shows a code editor with a CSS file named "styles.css" and a browser window titled "Preview CSS_table.html".

The CSS code in "styles.css" is:

```
# styles.css  x
1 table {
2   border: 1px solid;
3   font-size: 24px;
4   border-collapse: collapse;
5 }
6 th {
7   width:150px;
8 }
9
10 tr:nth-child(even) {
11   background-color: #gray;
12 }
13
14 tr td:nth-of-type(3) {
15   text-align: center;
16 }
17
18
```

The browser preview shows a table titled "Tableau des âges" with the following data:

| Prénom | Nom | Âge |
|----------|--------|-----|
| Rick | Grimes | 45 |
| Negan | | 50 |
| Glenn | Rhee | 28 |
| Maggie | Greene | 26 |
| Michonne | | 37 |
| Carl | Grimes | 15 |



Le site divtable.com/table-styler/ permet de construire visuellement des tableaux

On utilise l'événement **hover** :

```
# styles.css x
1  table {
2    border: 1px solid;
3    font-size: 24px;
4    border-collapse: collapse;
5  }
6
7  th {
8    width:150px;
9  }
10
11 tr td:nth-of-type(3) {
12   text-align: center;
13 }
14
15 tr:hover {background-color: #832b2b;}
```

The screenshot shows a code editor on the left and a preview window on the right. The code editor contains a CSS file named 'styles.css' with the following content:

```
# styles.css x
1  table {
2    border: 1px solid;
3    font-size: 24px;
4    border-collapse: collapse;
5  }
6
7  th {
8    width:150px;
9  }
10
11 tr td:nth-of-type(3) {
12   text-align: center;
13 }
14
15 tr:hover {background-color: #832b2b;}
```

The preview window shows a table titled "Tableau des âges" with the following data:

| Prénom | Nom | Âge |
|----------|--------|-----|
| Rick | Grimes | 45 |
| Negan | | 50 |
| Glenn | Rhee | 28 |
| Maggie | Greene | 26 |
| Michonne | | 37 |
| Carl | Grimes | 15 |

Disposition

RAPPEL : ELEMENTS "BLOCK" VS ELEMENTS "INLINE"

Un élément de niveau **block** démarre au début d'une ligne et occupe tout l'espace horizontal possible :

- <div>
- <h1>...<h6>
- <p>
- <form>
- <header>
- <footer>
- <section>
- ...

Un élément **inline** peut démarrer **n'importe où** et occupe seulement l'espace nécessaire :

-
- <a>
-
- ...

Disposition

PROPRIÉTÉ **display** POUR SURCHARGER UN COMPORTEMENT

La propriété **display** permet de spécifier si et comment doit être affiché un élément :

The screenshot shows a browser developer tools interface with three tabs:

- CSS_display.html**: Contains the following HTML code:

```
<link rel="stylesheet" href="styles.css">
<p>Des liens affichés comme des blocks :</p>
<a href="/html/default.asp">HTML</a>
<a href="/css/default.asp">CSS</a>
<a href="/js/default.asp">JavaScript</a>
```
- # styles.css**: Contains the following CSS rule:

```
a {
  display: block;
```
- Preview CSS_display.html**: Shows the rendered output where the links are displayed as separate blocks.

Output on the right: Des liens affichés comme des blocks :

[HTML](#)
[CSS](#)
[JavaScript](#)

The screenshot shows a browser developer tools interface with three tabs:

- CSS_display.html**: Contains the following HTML code:

```
<p>Une liste affichée en inline :</p>
<ul>
  <li><a href="/html/default.asp">HTML</a></li>
  <li><a href="/css/default.asp">CSS</a></li>
  <li><a href="/js/default.asp">JavaScript</a></li>
</ul>
```
- # styles.css**: Contains the following CSS rule:

```
li {
  display: inline;
```
- Preview CSS_display.html**: Shows the rendered output where the list items are displayed inline.

Output on the right: Une liste affichée en inline :

[HTML](#) [CSS](#) [JavaScript](#)

Disposition

VALEUR inline-block

Il existe une autre valeur pour la propriété **display : inline-block** :

- comparé à un **inline** : peut avoir une largeur, une hauteur, des marges et remplissages en haut et en bas
- comparé à un **block** : n'ajoute pas un saut de ligne après l'élément

Utilisation courante : barre de navigation horizontale ([exemple](#))

display: inline

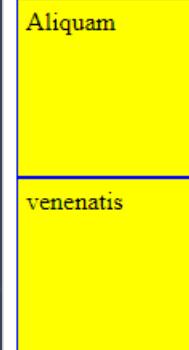
Lorem ipsum dolor sit amet, consectetur adipiscing elit. V erat volutpat. Aliquam venenatis gravida nisl sit amet.

display: inline-block

Lorem ipsum dolor sit amet, consectetur adipiscing elit. V erat volutpat. Aliquam venenatis gravida nisl sit amet.

display: block

Lorem ipsum dolor sit amet, consectetur adipiscing elit. V erat volutpat.



Disposition

WIDTH VS MAX-WIDTH

Pour empêcher un élément de niveau block de prendre toute la largeur de son conteneur, on peut utiliser **width**.

Avec la propriété **max-width**, l'élément devient adaptatif (*responsive*) :

```
3 <div class="ex1">Cet élément div a la  
4   propriété width</div>  
5 <br>  
6  
7 <div class="ex2">Cet élément div a les  
8   propriétés max-width et margin-auto</div>  
9  
10  
11  
12  
13  
14  
15  
16  
17
```

```
1 div {  
2   font-size: 20px;  
3 }  
4  
5 div.ex1 {  
6   width:600px;  
7   border: 3px solid #73AD21;  
8 }  
9  
10 div.ex2 {  
11   max-width:600px;  
12   margin: auto;  
13   border: 3px solid #73AD21;  
14 }  
15
```

Cet élément div a la propriété width

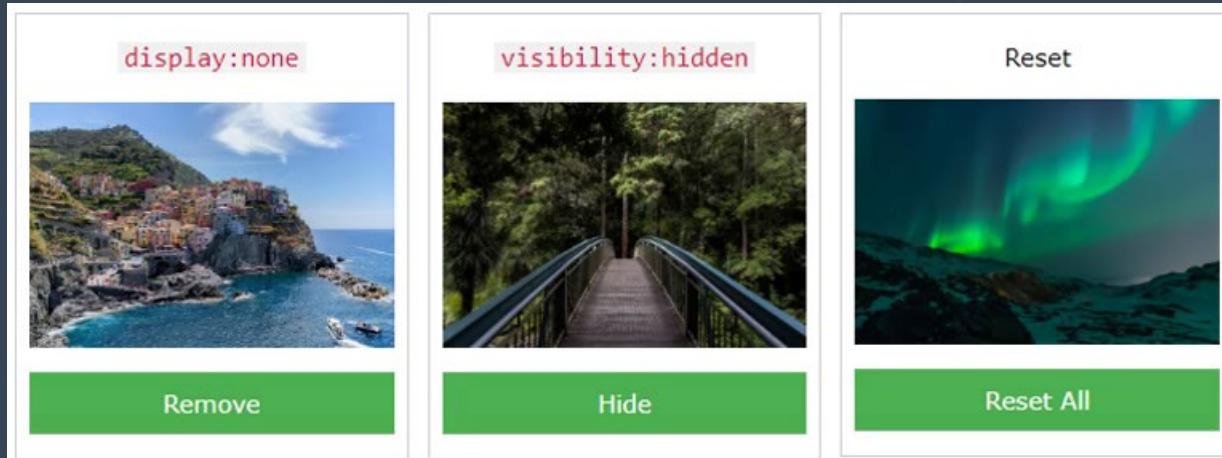
Cet élément div a les propriétés max-width et margin-auto

Disposition

CACHER UN ELEMENT

On peut facilement combiner CSS et JavaScript pour **cacher** des éléments soit :

- en **ne l'affichant pas** (propriété **display:none**) : l'élément n'occupe pas de place sur la page
- en **les rendant invisibles** (propriété **visibility:hidden**) : l'élément n'est pas affiché mais continue à occuper la place qu'il prendrait s'il était affiché



Autre exemple : créer une « info-bulle »

La propriété **position** spécifie la manière de positionner un élément ; 5 valeurs possibles :

| | |
|-----------------|--|
| <u>static</u> | valeur par défaut |
| <u>relative</u> | « à 30 px à gauche » de la position normale |
| <u>fixed</u> | « en bas à droite » de la fenêtre |
| <u>absolute</u> | « à 80 px du haut 20 px à droite » (du plus proche ancêtre non <i>static</i>) |
| <u>sticky</u> | en fonction de la barre de défilement |

```

3 <p>Le Lorem Ipsum est simplement du faux texte employé d
4   Ipsum est le faux texte standard de l'imprimerie dep
5   ensemble des morceaux de texte pour réaliser un livr
6
7 </p>
8
9 <div class="sticky">Je suis un pot de colle !!!</div>
10
11 <div style="padding-bottom:2000px">
12   <p>Contrairement à une opinion répandue, le Lorem Ip
13     racines dans une oeuvre de la littérature latine
14     Un professeur du Hampden-Sydney College, en Virg
15     consectetur, extrait d'un passage du Lorem Ipsum
16     classique, découvrit la source incontestable du
17     du "De Finibus Bonorum et Malorum" (Des Suprèmes
18     populaire pendant la Renaissance, est un traité
19     Ipsum, "Lorem ipsum dolor sit amet...", provien
20
21 </p>
22 </div>

```

```

1 div.sticky {
2   position: -webkit-sticky;
3   position: sticky;
4   top: 0;
5   padding: 5px;
6   background-color: #096409;
7   border: 2px solid #4CAF50;
8 }
9
10 p {
11   font-size:12pt;
12 }
13
14
15
16
17
18
19
20

```

Le Lorem Ipsum est simplement du faux texte employé dans la composition et la mise en page avant impression. Le Lorem Ipsum est le faux texte standard de l'imprimerie depuis les années 1500, quand un imprimeur anonyme assembla ensemble des morceaux de texte pour réaliser un livre spécimen de polices de texte.

Je suis un pot de colle !!!

Contrairement à une opinion répandue, le Lorem Ipsum n'est pas simplement du texte aléatoire. Il trouve ses racines dans une œuvre de la littérature latine classique datant de 45 av. J.-C., le rendant vieux de 2000 ans. Un professeur du Hampden-Sydney College, en Virginie, s'est intéressé à un des mots latins les plus obscurs, *consectetur*, extrait d'un passage du Lorem Ipsum, et en étudiant tous les usages de ce mot dans la littérature classique, a découvert la source incontestable du Lorem Ipsum. Il provient en fait des sections 1.10.32 et 1.10.33 du "De Finibus Bonorum et Malorum" (Des Suprêmes Biens et des Suprêmes Mauux) de Cicéron. Cet ouvrage, très populaire pendant la Renaissance, est un traité sur la théorie de l'éthique. Les premières lignes du Lorem Ipsum, "Lorem ipsum dolor sit amet...", proviennent de la section 1.10.32.

Disposition

ÉCRIRE SUR UNE IMAGE

```
<div class="container">  
    
  <div class="topleft">Haut Gauche</div>  
  <div class="topright">Haut Droite</div>  
  <div class="centered">Centre</div>  
  <div class="bottomleft">Bas Gauche</div>  
  <div class="bottomright">Bas Droite</div>  
</div>
```

```
1  div {  
2    color: red;  
3    font-size: 26px;  
4  }  
5  .container {  
6    position: relative;  
7  }  
8  
9  .centered {  
10   position: absolute;  
11   left: 0;  
12   top: 50%;  
13   width: 100%;  
14   text-align: center;  
15 }  
16  
17 .topleft {  
18   position: absolute;  
19   top: 8px;  
20   left: 16px;  
21 }  
22
```



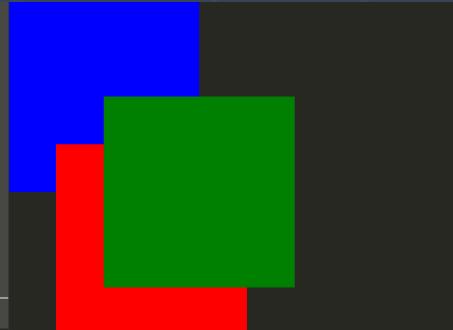
Les éléments sont positionnés selon **3 axes** !

La coordonnée selon l'axe z donne l'ordre d'empilement des éléments

Par défaut, un élément est positionné **par-dessus** un élément le précédent dans le code HTML. On utilise la propriété **z-index** pour modifier cet ordre :

```
<div id="div1"></div>
<div id="div2"></div>
<div id="div3"></div>
```

```
1 div {
2   position: absolute;
3   height: 200px;
4   width: 200px;
5 }
6
7 div#div1 {
8   z-index: 1;
9   left: 0px;
10  top: 0px;
11  background-color: blue;
12 }
13
14 div#div2 {
15   z-index: 3;
16   left: 100px;
17   top: 100px;
18   background-color: green;
19 }
20
21 div#div3 {
22   z-index: 2;
23   left: 50px;
24   top: 150px;
25   background-color: red;
26 }
```



- Centrer horizontalement du texte : `text-align: center`
- Centrer horizontalement un élément de type **block** : `margin: auto`
- Centrer horizontalement un élément de type **inline** : `display: block; margin-left: auto;`
`padding: value px;`
- Centrer verticalement :
- Aligner à droite ou à gauche : 2 possibilités : `position: absolute;` ou `float: left | right;`

⚠️ Un élément flottant sort du flux normal ! Il peut déborder de son conteneur et les éléments qui suivent sont mal positionnés

```
<div>
  
  Ici, l'image est flottante et plus grande
  que le conteneur, donc elle déborde * !!!
</div>

<p>
  Ce paragraphe devrait être sous l'image
</p>
```

```
1 □ div {
2   border: 3px solid #4CAF50;
3   padding: 5px;
4   font-size: 16pt;
5 }
6
7 □ img {
8   float: right;
9 }
10
11 □ p {
12   font-size: 16pt;
13 }
```

Ici, l'image est flottante et plus grande que le conteneur, donc elle déborde * !!!

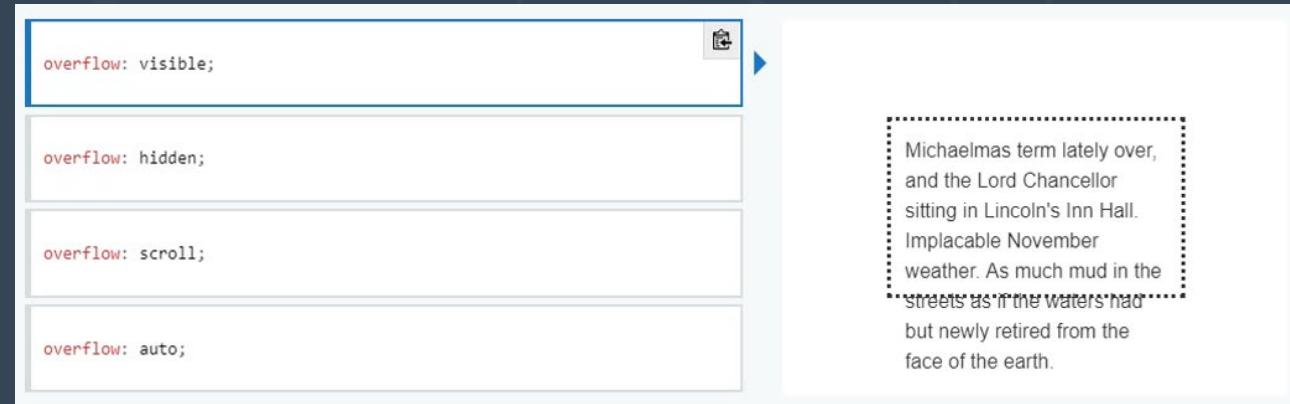
Ce paragraphe devrait être sous l'image !



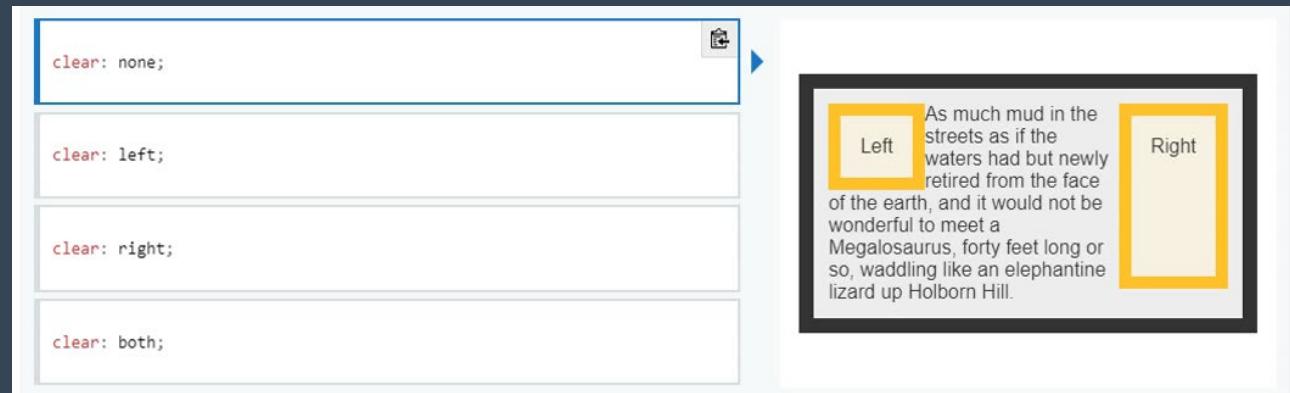
Disposition

PROBLEMES DES FLOTTANTS

Solution au problème de **débordement** : propriété **overflow** (comment gérer le débordement)



Solution au problème de **positionnement** : propriété **clear** (indique sous quels flottant placer l'élément)



Solution au problème des flottants :

```
<div>
  
Ici, l'image est flottante et plus grande que
le conteneur, donc elle déborde * !!!
</div>
```

```
<p>
Ce paragraphe a la propriété clear:right
</p>
```

```
<div class="clearfix">
  
  * ... sauf si on ajoute la propriété
  overflow: auto !
</div>
```

```
1  div {
2   border: 3px solid #4CAF50;
3   padding: 5px;
4   font-size: 16pt;
5  }
6
7  p {
8   clear:right;
9   font-size: 16pt;
10 }
11
12 img {
13   float: right;
14 }
15
16 .clearfix {
17   overflow: auto;
18 }
```

Ici, l'image est flottante et plus grande que le conteneur, donc elle déborde * !!!



Ce paragraphe a la propriété clear:right

* ... sauf si on ajoute la propriété overflow: auto !



Disposition

CREER UN MENU DE NAVIGATION

Les flottants sont (entre autres !) couramment employés pour créer des menus de navigation :

```
3  <nav>
4      <a href="#home" class="active">Home</a>
5      <a href="#news">News</a>
6      <a href="#contact">Contact</a>
7      <a href="#about">About</a>
8  </nav>
9
10
11
12
13
14
15
16
17
18
```

```
1  a {
2      float: left;
3      background-color: #333;
4      color: white;
5      text-align: center;
6      padding: 14px 16px;
7      text-decoration: none;
8  }
9
10 a:hover {
11     background-color: #0B0;
12 }
13
14 .active {
15     background-color: red;
16 }
17
```

Home News Contact About

Disposition

CREER UNE GRILLE

Avec tout ce qu'on a vu, on a tous les outils pour créer une grille d'éléments simple :



```
<div class="box" style="background-color: #bbb">  
    <p>Un paragraphe dans une boîte</p>  
</div>  
<div class="box" style="background-color: #ccc">  
    <p>Un paragraphe dans une boîte</p>  
</div>  
<div class="box" style="background-color: #ddd">  
    <p>Un paragraphe dans une boîte</p>  
</div>  
</div>  
  
<p>  
    Le Lorem Ipsum est simplement du faux texte employé dans la composition et la mise en page avant impression. Le Lorem Ipsum est le faux texte standard de l'imprimerie depuis les années 1500, quand un imprimeur anonyme assembla ensemble des morceaux de texte pour réaliser un livre. Il n'a pas de sens littéraire, mais il utilise des mots de la langue latine pour générer un document qui ressemble à une page imprimée. Il a été utilisé dans les types de caractères et les logiciels de composition depuis les années 1980.  
</p>
```

```
1  * {  
2  box-sizing: border-box;  
3  color: tomato;  
4  font-size: 16pt;  
5  }  
6  
7 .box {  
8  float: left;  
9  width: 33.33%;  
10 padding: 50px;  
11 }  
12  
13 .clearfix::after {  
14  content: "";  
15  clear: both;  
16  display: table;  
17 }
```

Le Lorem Ipsum est simplement du faux texte employé dans la composition et la mise en page avant impression. Le Lorem Ipsum est le faux texte standard de l'imprimerie depuis les années 1500, quand un imprimeur anonyme assembla ensemble des morceaux de texte pour réaliser un livre. Il n'a pas de sens littéraire, mais il utilise des mots de la langue latine pour générer un document qui ressemble à une page imprimée. Il a été utilisé dans les types de caractères et les logiciels de composition depuis les années 1980.



Les boîtes ont un **padding** ; il faut en tenir compte sinon les 3 boîtes ne tiennent pas (pourquoi ?)

Exemple avec des images : www.w3schools.com/css/css_float.asp



Alternative aux flottants et aux frameworks (Bootstrap...) pour créer des sites *responsives*

Principe : créer un parent (ou *conteneur*) avec la propriété `display: flex`

```
<!DOCTYPE html>
<html>
<head>
<style>
.flex-container {
  display: flex;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  margin: 10px;
  padding: 20px;
  font-size: 30px;
}
</style>
</head>
<body>

<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

<p>A Flexible Layout must have a parent element with the <em>display</em> property set to <em>flex</em>.</p>

<p>Direct child elements(s) of the flexible container automatically becomes flexible items.</p>

</body>
</html>
```



A Flexible Layout must have a parent element with the `display` property set to `flex`.

Direct child elements(s) of the flexible container automatically becomes flexible items.

? Alternative aux flottants et aux frameworks (Bootstrap...) pour créer des sites *responsive*s

Principe : créer un parent (ou *conteneur*) avec la propriété `display:flex`

6 propriétés :

- `flex-direction` : column, column-reverse, row, row-reverse
- `flex-wrap` : wrap, nowrap, wrap-reverse
- `flex-flow` : raccourci pour `flex-direction` `flex-wrap`
- `justify-content` : flex-start, center, flex-end, space-around, space-between
- `align-items` (alignement vertical) : stretch, flex-start, center, flex-end, baseline
- `align-content` (alignement vertical) : space-between, space-around, stretch, center,
flex-start, flex-end

💡 Solution idéale pour centerer parfaitement un élément

💡 Les enfants d'un conteneur flex deviennent automatiquement flex

- On peut changer l'**ordre d'affichage** avec la propriété **order** :

```
<div class="flex-container">
  <div style="order: 3">1</div>
  <div style="order: 2">2</div>
  <div style="order: 4">3</div>
  <div style="order: 1">4</div>
</div>
```

Use the order property to sort the flex items as you like:



- La propriété **flex** est un raccourci pour flex-grow, flex-shrink, flex-basis

```
/* Create two unequal columns that sits next to each other */
/* Sidebar/left column */
.side {
  flex: 30%;
  background-color: #f1f1f1;
  padding: 20px;
}

/* Main column */
.main {
  flex: 70%;
  background-color: white;
  padding: 20px;
}
```



Module *Grid Layout*

Permet de construire facilement des dispositions sous forme de grilles : **display: grid**

The screenshot shows a code editor with three tabs: `grid.html`, `grid.css`, and `Preview grid.html`.

grid.html:

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <link rel="stylesheet" href="grid.css">
6 </head>
7
8 <body>
9
10 <h1>Grid Elements</h1>
11
12 <p>A Grid Layout must have a parent element with <b>display: grid</b> set.
13
14 <p>Direct child element(s) of the grid container must have <b>grid-item</b> class.
15
16 <div class="grid-container">
17   <div class="grid-item">1</div>
18   <div class="grid-item">2</div>
19   <div class="grid-item">3</div>
20   <div class="grid-item">4</div>
21   <div class="grid-item">5</div>
22   <div class="grid-item">6</div>
23   <div class="grid-item">7</div>
24   <div class="grid-item">8</div>
25   <div class="grid-item">9</div>
26 </div>
27
28 </body>
29
30 </html>
```

grid.css:

```
1 .grid-container {
2   display: grid;
3   grid-template-columns: auto auto auto;
4   background-color: #2196F3;
5   padding: 10px;
6 }
7
8 .grid-item {
9   background-color: rgba(255, 255, 255, 0.8);
10  border: 1px solid rgba(0, 0, 0, 0.8);
11  padding: 20px;
12  font-size: 30px;
13  text-align: center;
14 }
```

Preview grid.html:

Grid Elements

A Grid Layout must have a parent element with the *display* property set to *grid* or *inline-grid*.

Direct child element(s) of the grid container automatically becomes grid items.

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Attention à la compatibilité avec les navigateurs anciens !

Module *Grid Layout*

- Définition et largeur des colonnes : `grid-template-columns: 200px auto auto auto;`
- Définition et largeur des lignes : `grid-template-rows: 80px auto 200px;`
- Espacer les éléments de la grille : `grid-column-gap`, `grid-row-gap` et `grid-gap`
- Fusionner des cellules : `grid-column-start: 1; grid-column-end: 3;` ou `grid-column: 1 / 3;` ou encore `grid-column: 1 / span 2;` (idem avec `grid-row*`)
- Crée une *zone* : voir https://www.w3schools.com/css/tryit.asp?filename=trycss_grid_grid-area2
- Gérer l'espace autour des colonnes (`justify-content`) ou vertical (`align-content`) : `space-evenly` | `space-around` | `space-between` | `center` | `start` | `end`

Références :

- css-tricks.com/snippets/css/complete-guide-grid
- www.w3schools.com/css/css_grid_container.asp

Flexbox vs. Grid Layout

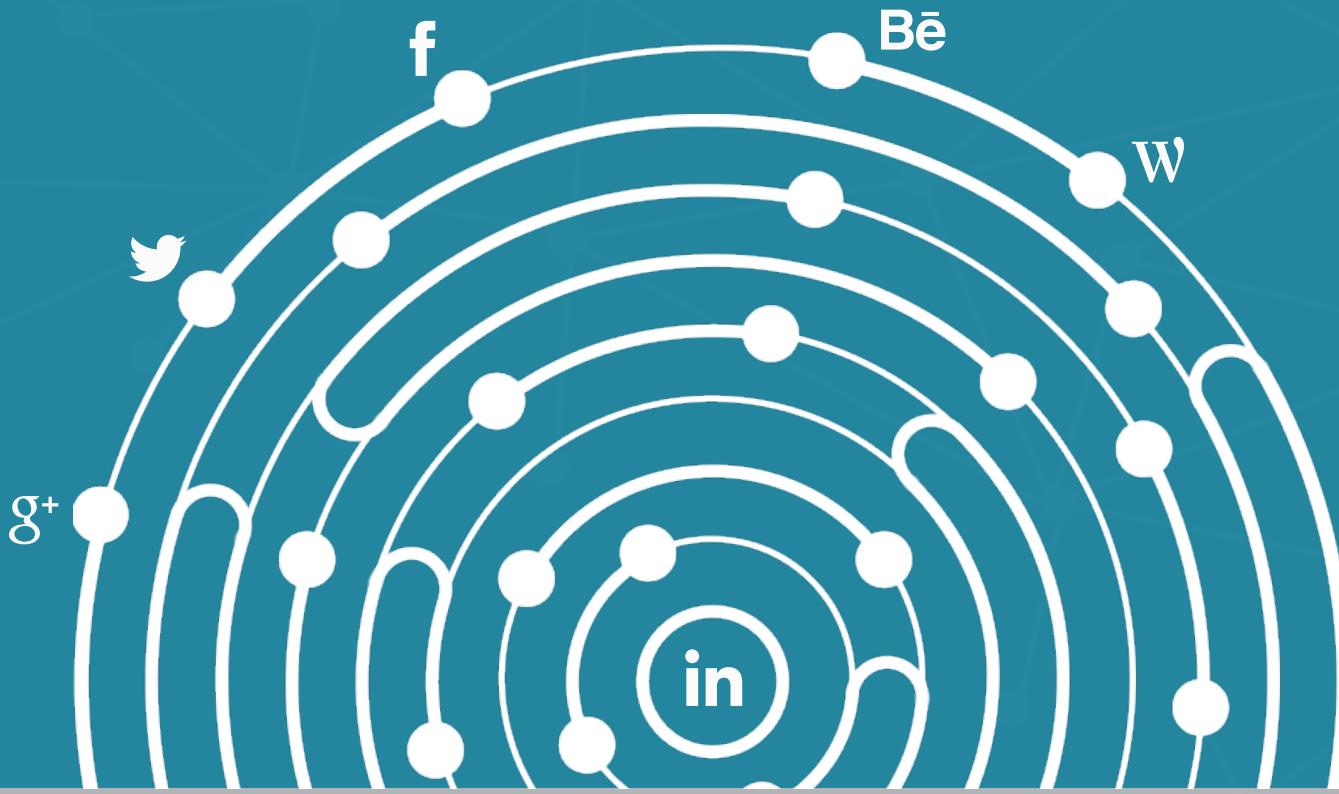
- Flexbox
 - fonctionne sur *une* dimension (soit horizontalement, soit verticalement) (mais peut déborder (*wrap*) sur plusieurs lignes si l'espace est insuffisant)
- CSS Grid
 - permet de travailler selon deux axes, et permet de positionner précisément des éléments dans les cellules définies par des lignes et des colonnes

💡 Chacune des deux technologies permet des choses que ne permet pas l'autre

Flexbox vs. Grid Layout

Règle :

- Flexbox est *orienté contenu* (les règles CSS sont surtout au niveau des enfants) ; on l'utilise pour :
 - se concentrer sur l'écoulement du contenu (content flow) plutôt que sur le placement ; les éléments flex peuvent grossir et rétrécir en fonction de leur contenu et de l'espace disponible
 - aligner des éléments
- CSS Grid est *orienté conteneur* (les règles CSS sont surtout au niveau de l'élément parent) ; on l'utilise pour :
 - se concentrer sur le placement des éléments
 - les mises en page plus complexes



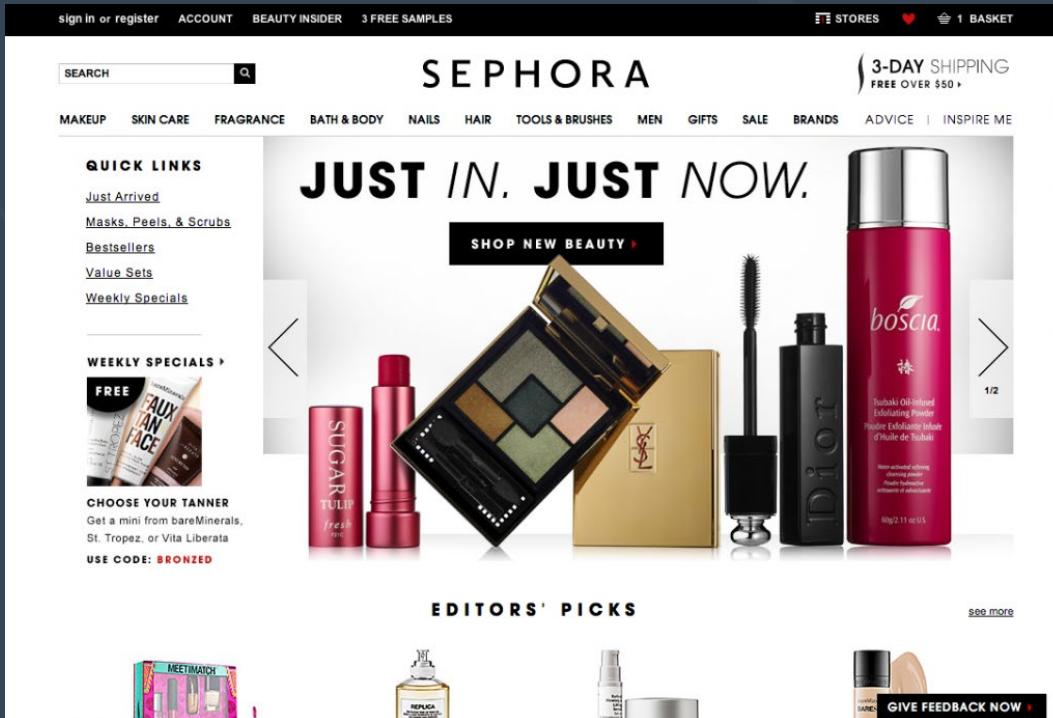
Responsive Web Design (RWD)

Site responsive vs. site mobile

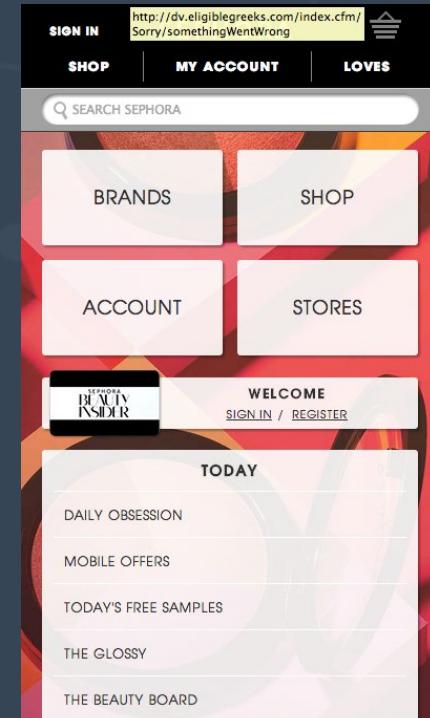
2 APPROCHES POUR UN MÊME PROBLÈME

2 manières de créer un site web consultable sur PC, tablette, smartphone... :

1. créer une version pour PC, et une version **spécialement pour mobile** (ou *site m.*)



www.sephora.com



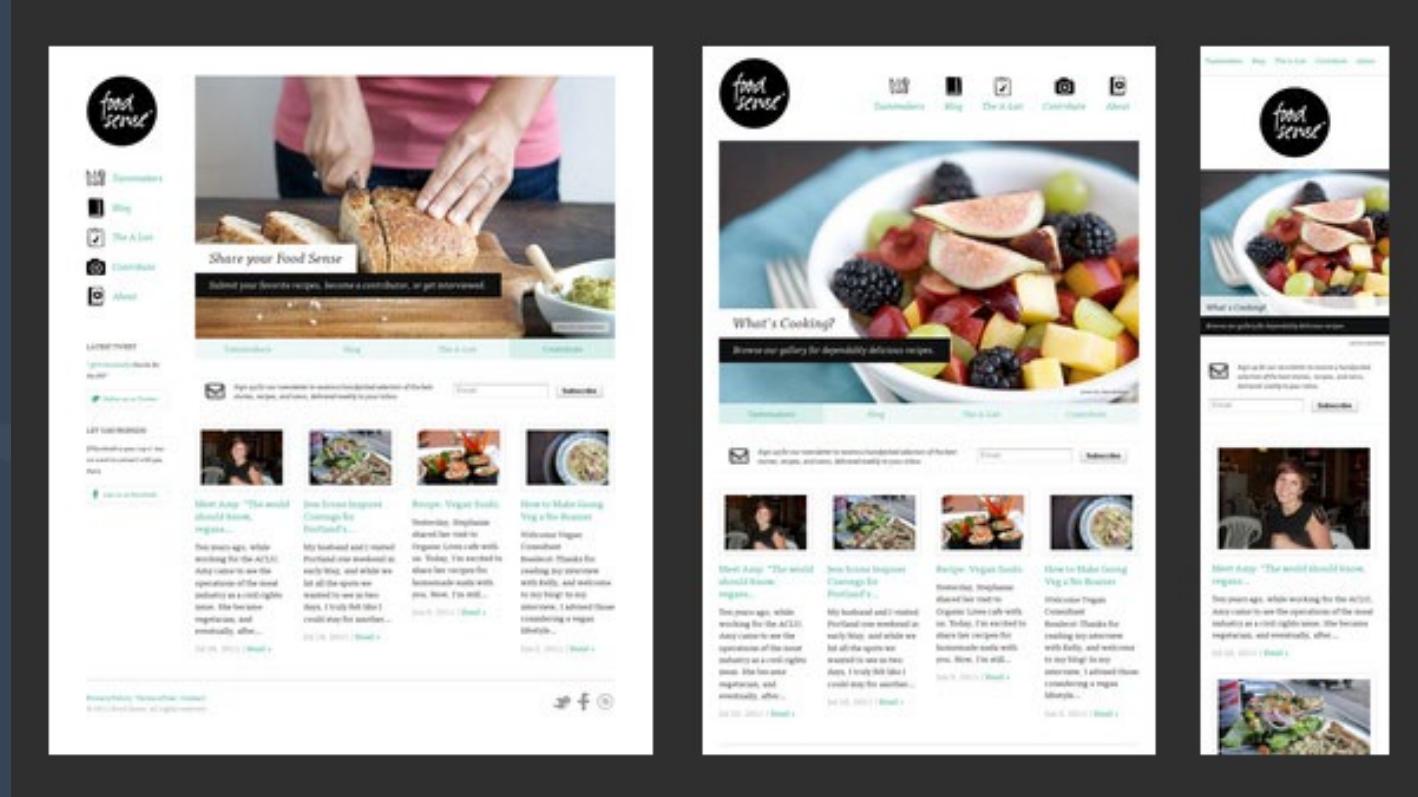
m.sephora.com

Site responsive vs. site mobile

2 APPROCHES POUR UN MÊME PROBLÈME

2 manières de créer un site web consultable sur PC, tablette, smartphone... :

2. créer un site qui s'adapte *automatiquement* à la largeur de l'écran



Source : <https://www.eeweefr.le-responsive-web-design-rwd-cest-quoi/>

Site responsive vs. site mobile

2 APPROCHES POUR UN MÊME PROBLÈME

| | Avantages | Inconvénients |
|-------------|---|--|
| Site mobile | <ul style="list-style-type: none">- offrir une approche personnalisée selon la cible- référencement adapté au contenu de chaque support- Interconnexion avec les fonctions du téléphone (GPS, NFC...) | <ul style="list-style-type: none">- coût de gestion dupliqué- création du contenu dupliquée- « concurrence » des applications |
| RWD | <ul style="list-style-type: none">- toucher un maximum d'internautes- une seule plateforme à développer et maintenir- Google favorise les sites RWD | <ul style="list-style-type: none">- plus complexe à mettre en place- usages différents sur mobile et desktop- problématique du contenu (quantité vs densité)- temps de chargement des éléments cachés |

- **Surface physique** : nombre physique de pixels sur l'écran (= définition ou *resolution*)

iPhone 3 : 320 x 480 px

iPhone 4 : 640 x 960 px

iPhone 11 Pro Max : 1242 x 2688 px

- **Surface utilisable (« pixels CSS »)** : nombre de pixels virtuels que l'appareil pense avoir, et sur lequel il fonde son affichage

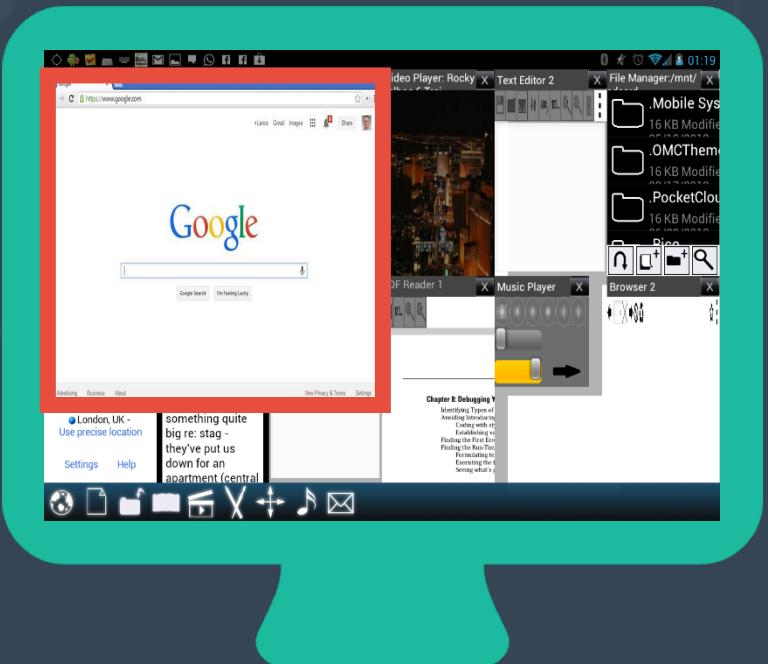
iPhone 3 : 320 x 480 px

iPhone 4 : 320 x 480 px

iPhone 11 Pro Max : 414 x 896 px

Problème : un pixel CSS n'est pas toujours égal à un pixel physique !

Viewport = surface de la fenêtre du navigateur



Viewport sur un ordinateur de bureau : OK



Viewport sur un smartphone ?

Pas de « fenêtre », pas de barres de défilement... !



Problème : le *viewport* d'un appareil mobile ne correspond... ni à la taille réelle de l'écran... ni à la taille en pixels CSS !

En fait, la valeur initiale du *viewport* ne dépend pas du terminal, mais du **navigateur mobile**

- Chrome, Opera mobile, Safari : **980 px**
- Internet Explorer mobile : **1024 px**



La valeur initiale du *viewport* est généralement supérieure à la surface physique

⇒ permet d'afficher une page web d'une largeur de 980 px sur un écran de 640px en appliquant un « dézoom » :

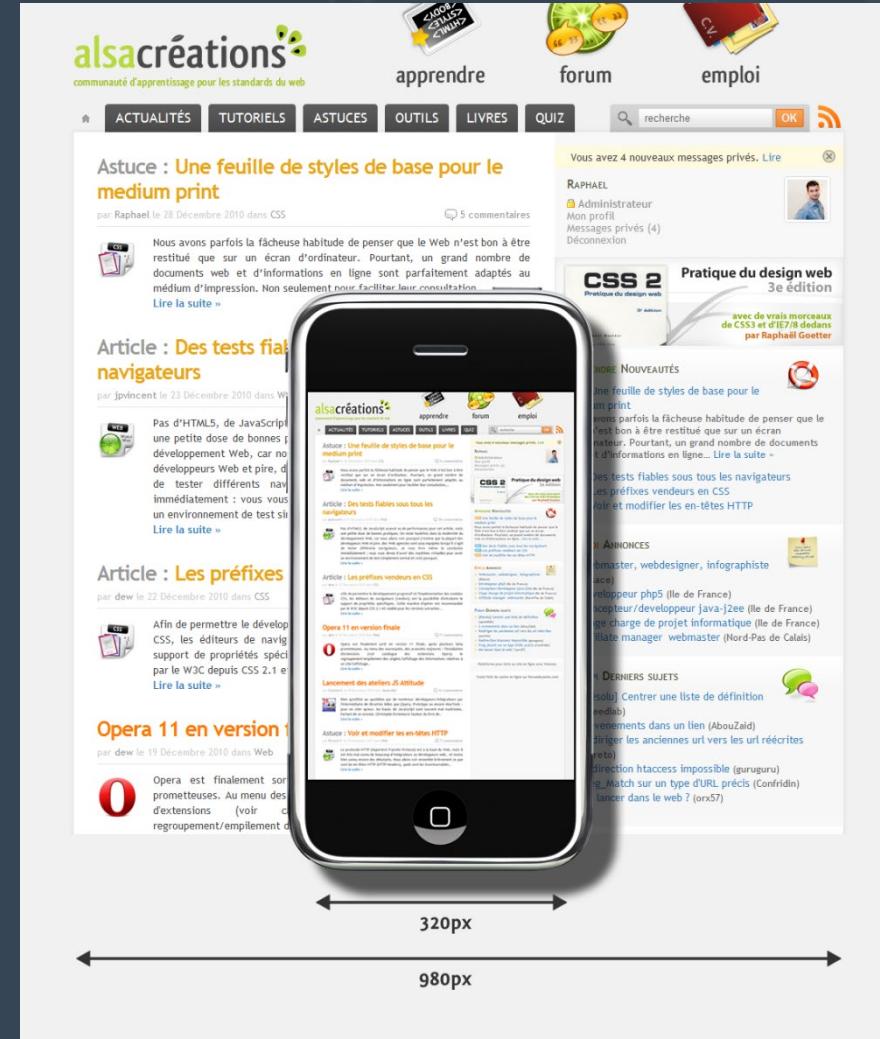
$$\text{zoom} = \text{device-width} / \text{viewport}$$

où **device-width** désigne la largeur CSS

Web mobile

UNE NOTION IMPORTANTE : LE VIEWPORT

Exemple :



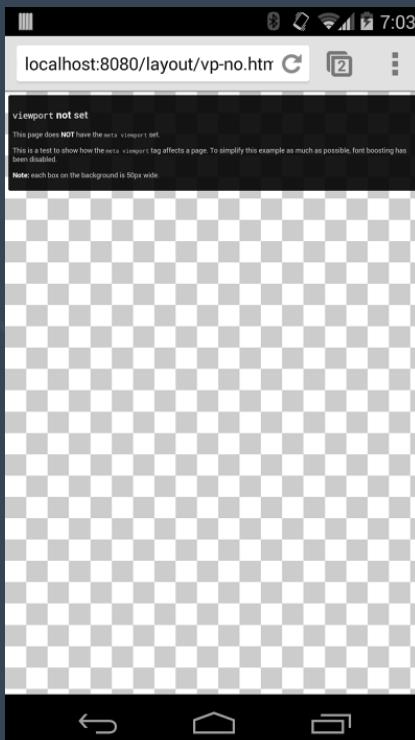
Viewport

SOLUTION : LA BALISE <meta> VIEWPORT

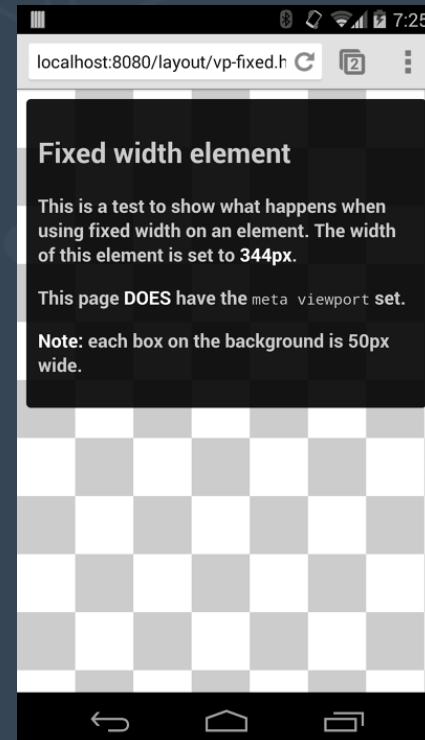


Il est possible d'imposer la taille du viewport... avec une balise HTML :

```
<meta name="viewport" content="width=360px">
```



Sans viewport



Avec viewport

Viewport

SOLUTION : LA BALISE <meta> VIEWPORT

💡 Il est possible d'imposer la taille du viewport... avec une balise HTML :

```
<meta name="viewport" content="width=360px">
```

▪ **device-width** permet d'être indépendant de l'appareil :

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

▪ **initial-scale** : niveau de zoom initial (*utile sur certains appareils en mode paysage*)

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

💡 Le W3C a intégré cette règle en CSS, qui devrait à terme remplacer la balise <meta name="viewport"> :

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

- CSS 2 : *media types* ⇒ la règle `@media` permet de définir différents styles pour différents types de media
 - un ensemble de règles pour les écrans d'ordinateurs
 - un ensemble de règles pour les imprimantes
 - un ensemble de règles pour les dispositifs portables

⇒ malheureusement, peu de support (à part pour les imprimantes)
- CSS 3 : *media queries* ⇒ on regarde les capacités d'un appareil plutôt que son type
 - largeur / hauteur de l'écran
 - largeur / hauteur de la fenêtre (*viewport*)
 - orientation...

Media Queries

POUR CREER DES SITES ADAPTATIFS (*RESPONSIVE*)

Syntaxe :

```
@media not | only mediatype and (expressions) {  
    CSS-Code;  
}
```

Media types CSS3 : all, print, screen, speech

Exemple 1 :

The screenshot shows a browser's developer tools with the CSS panel open. The code editor contains the following CSS:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: pink;
}

@media screen and (min-width: 480px) {
    body {
        background-color: lightgreen;
    }
}
```

At the top of the browser window, there are icons for home, refresh, and search, followed by a green "Run" button. To the right of the run button, it says "Result Size: 524 x 227". On the right side of the browser window, there is a large green box containing the text:

Resize the browser window to see the effect!

The media query will only apply if the media type is screen and the viewport is 480px wide or wider.

Media Queries

POUR CREER DES SITES ADAPTATIFS (*RESPONSIVE*)

Exemple 2 :

The screenshot shows a browser window with developer tools open. On the left, the CSS panel displays the following code:

```
<style>
* {
  box-sizing: border-box;
}

/* Create four equal columns that floats next to each other */
.column {
  float: left;
  width: 25%;
  padding: 20px;
}

/* Clear floats after the columns */
.row:after {
  content: "";
  display: table;
  clear: both;
}

/* On screens that are 992px wide or less, go from four columns to two columns */
@media screen and (max-width: 992px) {
  .column {
    width: 50%;
  }
}

/* On screens that are 600px wide or less, make the columns stack on top of each other instead of next to each other */
@media screen and (max-width: 600px) {
  .column {
    width: 100%;
  }
}
</style>
```

On the right, the preview area shows a "Responsive Four Column Layout" with the title "Result Size: 902 x 640". It contains four columns labeled "Column 1", "Column 2", "Column 3", and "Column 4", each containing placeholder text "Some text..". A note at the top of the preview says: "Resize the browser window to see the responsive effect. On screens that are 992px wide or less, the columns will resize from four columns to two columns. On screens that are 600px wide or less, the columns will stack on top of each other instead of next to each other."

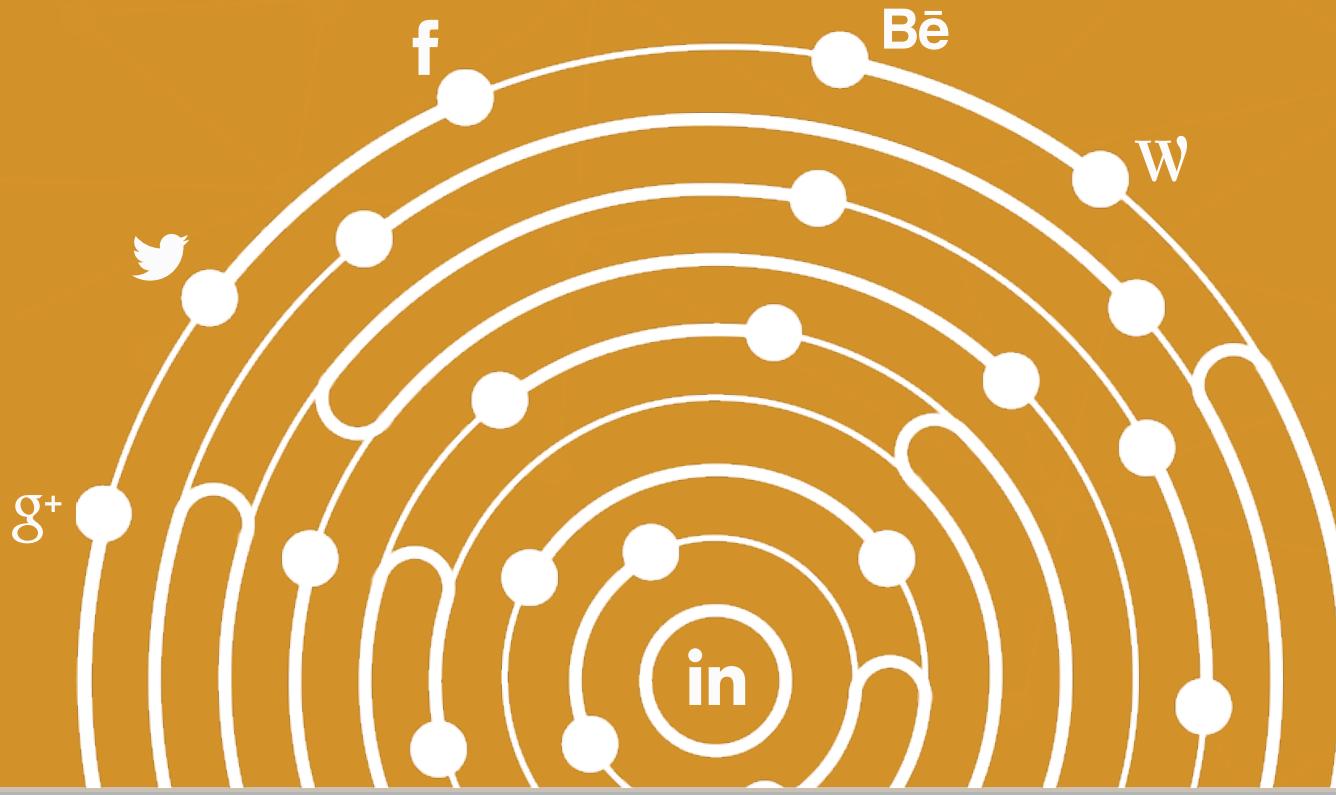
D'autres exemples : www.w3schools.com/css/css3_mediaqueries_ex.asp

Effets supplémentaires

- Opacité au survol de la souris : www.w3schools.com/css/css_image_transparency.asp
- Barres de navigation : www.w3schools.com/css/css_navbar.asp
- Menus déroulants : www.w3schools.com/css/css_dropdowns.asp
- Dispositions diverses : www.w3schools.com/css/css_website_layout.asp
- Transformations géométriques : soit en 2D soit en 3D
- Animations : https://www.w3schools.com/css/css3_animations.asp



Des questions ?



3^{ème} partie : Bases de JavaScript

Encore un langage !?!

- HTML est le langage de définition du **contenu** d'une page web
- CSS est le langage de **mise en forme** de cette page web
- Avec JavaScript vous allez pouvoir **programmer le comportement** de votre page web !

JavaScript est le langage de programmation du web*

JavaScript est (assez) facile à apprendre

JavaScript n'a (presque) RIEN à voir avec Java**

*Enfin... il y en a d'autres !

**A part avoir pompé le nom...



Ca sort d'où ?

- 23 janvier 1993 : sortie de NCSA Mosaic ([Eric Bina, Marc Andreessen](#)) ; premier navigateur qui affiche les images et les formulaires
- Avril 1994 : M. Andreessen et Jim Clark créent [Mosaic Communication Corp.](#) : première entreprise à miser sur le net
- Octobre 1994 : sortie de [Mosaic Netscape 0.9](#) (première version à supporter les [cookies](#)), qui deviendra [Netscape Navigator](#) ; [Mosaic Corp.](#) devient [Netscape](#)
- 1995 – 1998 : première guerre des navigateurs (Microsoft vs. Netscape)
- Janvier 1998 : lancement du projet open-source [Mozilla](#) (Mosaic + Godzilla)
- 2003 : dissolution de [Netscape](#)



Ca sort d'où ?

- 1995 : en pleine guerre des navigateurs, Netscape réalise que le web doit être plus **dynamique**
- Septembre 1995 : **Brendan Eich** (Netscape) écrit en 10 jours *LiveScript*, rebaptisé ensuite **JavaScript**
- Mars 1996 : sortie officielle de **JavaScript 1.0**
- Août 1996 : **Microsoft** riposte : sortie de **JScript**
 - ⇒ implémentations différentes ⇒ développement web difficile ⇒ adoption de JavaScript ralentie
- Novembre 1996 : **Netscape** soumet **JavaScript** à **ECMA International** pour standardisation
- 1998 – 2003 : **ECMAScript 1, 2 et 3**
- 2003 : Microsoft essaie de stopper l'évolution de JavaScript ; le projet est au point mort
- 2005 : **Jesse Garrett** décrit **AJAX**, un ensemble de technologies dont JavaScript est l'épine dorsale ; de nombreuses initiatives (jQuery, Prototype, Dojo...) voient le jour
- Décembre 2009 : la hache de guerre est enterrée ; sortie de **ECMAScript 5**



Ca sort d'où ?

JavaScript vs ECMAScript vs ES

| Date | Version |
|---------------|----------------------------------|
| Mars 1996 | JavaScript 1.0 |
| Juin 1997 | ECMAScript 1.0 |
| Juin 1998 | ECMAScript 2.0 / JavaScript 1.3 |
| Décembre 1999 | ECMAScript 3.0 |
| Novembre 2000 | JavaScript 1.5 |
| Décembre 2009 | ECMAScript 5 |
| Juillet 2010 | JavaScript 1.8.5 |
| Juin 2015 | ECMAScript 2015 = ES2015 = ES 6 |
| Juin 2016 | ECMAScript 2016 = ES2016 = ES 7 |
| Juin 2017 | ECMAScript 2017 = ES2017 = ES 8 |
| Juin 2018 | ECMAScript 2018 = ES2018 = ES 9 |
| Juin 2019 | ECMAScript 2019 = ES2019 = ES 10 |
| Juin 2020 | ECMAScript 2020 = ES2020 = ES 11 |

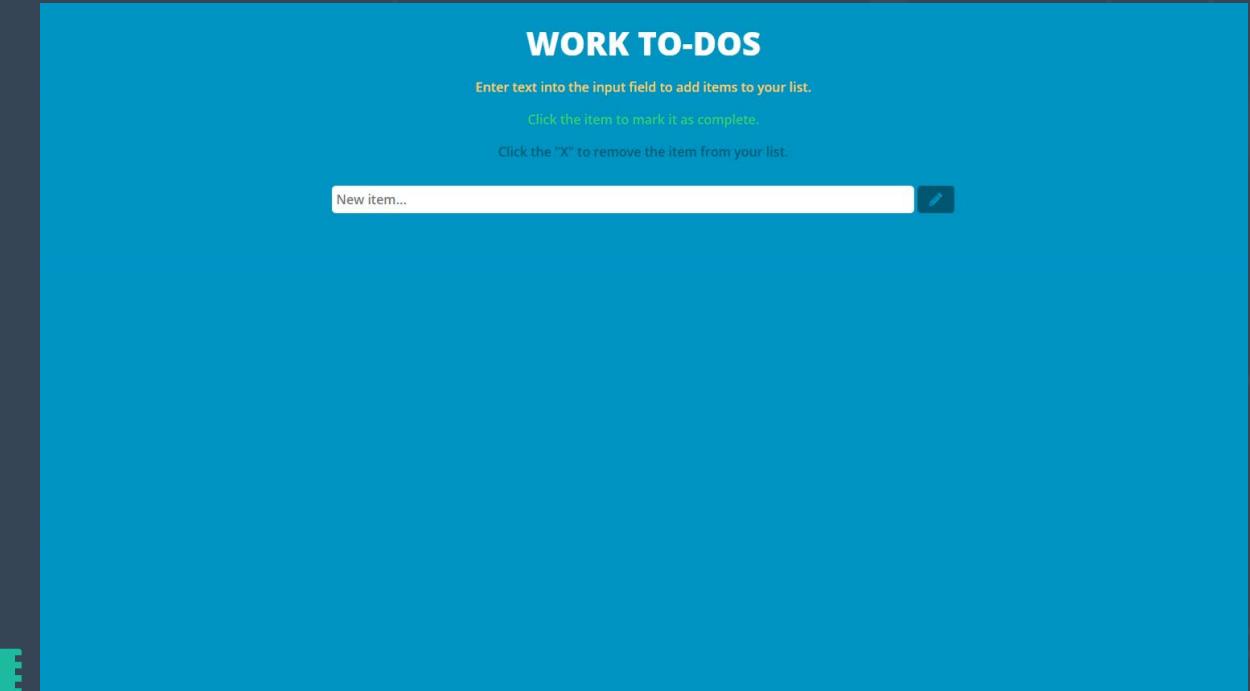
Ca sert à quoi ?

JavaScript peut :

- modifier le **contenu** HTML (ex. : le texte d'un paragraphe)
- modifier la **valeur** d'un attribut HTML (ex. : la source d'une image)
- modifier le **CSS**
- **afficher** ou **cacher** des éléments

Quelques exemples :

- <https://codepen.io/JohnPaulFich/pen/MXmzzM>
- <https://codepen.io/nswamy14/pen/MXoZjg?page=1>
- <https://codepen.io/aomyers/pen/LWOwpR>



JavaScript :

- est un langage de programmation *multi-paradigmes* : script, orienté objet à prototype, impératif et fonctionnel
- est un langage de programmation *interprété*
- permet de rendre les pages web *interactives* et a permis les *web apps*
- est utilisé *côté client ET côté serveur* (JScript / ASP depuis longtemps, Node.js plus récemment)
- est un langage à *typage dynamique* (on verra un peu plus loin de quoi il s'agit)
- est sensible à la *casse*
- évolue constamment !

Une bonne adresse : <https://fr.javascript.info/>

JavaScript

BALISE <script>

- Le code JavaScript s'insère entre les balises <script> et </script>
 - soit directement du code JavaScript
 - soit un [lien vers un fichier](#) contenant le code (préférable)
- On peut avoir autant de scripts qu'on veut dans un document HTML
- On peut insérer un script dans les sections **head** et **body**
 - 💡 Souvent à la fin du **body** pour accélérer le chargement de la page
- Pensez à **commenter** votre code ! (avec // ou /* ... */)
- Pensez aux **outils de débogage** offerts par les navigateurs (console...) !

maPage.html

```
<head>
  <script src='monScript.js'></script>
</head>

<body onload="my_function()">
  <p id="mon_paragraphe">
    Salut le monde !
  </p>
</body>
```

monScript.js

```
function my_function() {
  document.getElementById("mon_paragraphe")
    .innerHTML = "Hello World!";
}
```

⇒ Le contenu « Salut le monde ! » est remplacé par « Hello World! »

Déclaration des variables

INSTRUCTION LET

- L'instruction **let** : la variable est limitée à celle du *bloc courant* (limité par des accolades)

```
function test(){
  if(true){
    let maVar = 10;
    console.log(maVar); // 10
  }

  console.log(maVar); // Erreur, maVar n'existe pas hors du bloc "if"
}
```

Rem. : une variable déclarée hors de tout bloc est *globale*

Rem. : une variable peut être initialisée *après* avoir été déclarée :

```
let maVar; // undefined
maVar = 10;
```



Déclaration des variables

INSTRUCTION CONST

- L'instruction **const** : la variable **ne peut plus être modifiée** après avoir été initialisée

```
const PI = 3.14159;  
const LANG = 'fr';
```

```
const CONFIG = true;  
CONFIG = false;           // Erreur : on ne peut pas modifier une constante
```

Rem. : par convention, les noms de variables constantes sont en *majuscules*

Rem. : une constante doit être initialisée *lors de sa déclaration*



Déclaration des variables

INSTRUCTION VAR

- L'instruction **var** est similaire à **let**, mais la variable a une portée de *fonction*, quel que soit le bloc où elle a été déclarée :

```
function test(){
  if(true){
    var maVar = 10;
    console.log(maVar); // 10
  }

  console.log(maVar); // ok, la variable existe dans toute la fonction
}
```

Rem. : une variable déclarée hors d'une fonction est *globale*

Rem. : une variable déclarée sans mot-clé revient à créer une variable globale :

```
maVar = 5; // maVar est globale
```



JavaScript est un langage à typage dynamique :

- pas de **int**, **char**, **bool**... pour *déclarer* des variables (mais les variables sont typées)
- juste les mots **let** (portée de bloc), **var** (portée de fonction) et **const** (constante)

Le type est évalué *dynamiquement*, en fonction du contenu, et de gauche à droite :

- **let x;** // x est *undefined*
- **x = 5;** // x est maintenant un *number*
- **x = "Hello";** // x est maintenant un *string*
- **let var1 = 16 + 4 + 'voiture';** // var1 = "20voiture"
- **let var2 = "voiture" + 16 + 4;** // var2 = "voiture164"
- **let z = + "5";** // z = 5

Connaître le type : **typeof()**

© GREGORY MOREL

Les chaînes peuvent être initialisées avec des guillemets simples ou doubles

TECHNIQUES ET LANGAGES DU WEB – 2022-2023

Opérateurs arithmétiques et logiques

| Opérateur arithmétique | Description |
|------------------------|---|
| + | Addition |
| - | Soustraction |
| * | Multiplication |
| ** | Exponentiation (ES2016) |
| / | Division |
| % | Modulo (Reste de la division) |
| ++ | Incrément |
| -- | Décrément |

| Opérateur logique | Description |
|---------------------|-------------------------------|
| && | ET logique |
| | OU logique |
| ! | Négation |
| Opérateur bit à bit | Description |
| & | ET |
| | OU |
| ~ | NON |
| ^ | OU EXCLUSIF |
| <<, >> | Décalage à gauche ou à droite |

Opérateurs d'affectation

| Opérateur | Exemple | Équivalent à |
|-----------|-----------|--------------|
| = | $x = y$ | $x = y$ |
| $+=$ | $x += y$ | $x = x + y$ |
| $-=$ | $x -= y$ | $x = x - y$ |
| $*=$ | $x *= y$ | $x = x * y$ |
| $/=$ | $x /= y$ | $x = x / y$ |
| $%=$ | $x %= y$ | $x = x \% y$ |
| $**=$ | $x **= y$ | $x = x ** y$ |



Opérateurs de comparaison

| Opérateur | Description |
|--------------------|---|
| <code>==</code> | égal à |
| <code>===</code> | valeur égale et type identique |
| <code>!=</code> | Différent |
| <code>!==</code> | valeur différente ou type différent |
| <code>></code> | supérieur à |
| <code><</code> | Inférieur à |
| <code>>=</code> | supérieur ou égal à |
| <code><=</code> | Inférieur ou égal à |
| <code>?</code> | opérateur ternaire (ex. : <code>a > b ? console.log(a) : console.log(b)</code>) |

Conditions

Syntaxe similaire à Java, C++... :

```
var greeting;  
  
var time = new Date().getHours();  
  
if (time < 10) {  
  
    greeting = "Good morning";  
  
} else if (time < 20) {  
  
    greeting = "Good day";  
  
} else {  
  
    greeting = "Good evening";  
  
}
```

Opérateurs logiques : `&&`, `||`, `!`

Opérateurs de comparaison : `==`, `===(même valeur ET même type)`, `!=`, `!==`, `<`, `>=...`

- Pop / Push : enlever / ajouter un élément **à la fin** du tableau

```
var cars = ["Renault", "Volvo", "BMW"];
var lastElem = cars.pop();          // lastElem contient "BMW", cars contient ["Renault", "Volvo"]
cars.push("Tesla");                // cars contient ["Renault", "Volvo", "Tesla"]
```

- Shift / Unshift : enlever / ajouter un élément **au début** du tableau

```
var firstElem = cars.shift();      // firstElem contient "Renault", cars contient ["Volvo", "Tesla"]
cars.unshift("Ferrari");           // cars contient ["Ferrari", "Volvo", "Tesla"]
```

 **push()** et **unshift()** renvoient le nombre d'éléments présents désormais dans le tableau

- **delete cars[k]** supprime un élément donné mais **laisse son emplacement à empty**

- Pour ajouter ou supprimer un élément au milieu d'un tableau on utilisera **splice()** :

```
var fruits = ["banane", "orange", "pomme", "mangue", "ananas", "cerise"];
fruits.splice(2, 3);      // on supprime 3 éléments à partir de l'indice 2
                         // fruits contient maintenant ["banane", "orange", "cerise"]
```

```
fruits.splice(2, 0, "fraise", "poire");
// on supprime 0 éléments à partir de l'indice 2 et on ajoute 2 éléments
// fruits contient maintenant ["banane", "orange", "fraise", "poire", "cerise"]
```

- Pour concaténer deux tableaux : **tab1.concat(tab2)** (laisse les tableaux existants intacts)
- Pour copier un tableau à partir de l'indice **i** : **slice(i)** (laisse le tableau existant intact)



Tableaux : tri

- Tri : `cars.sort()` // le tableau est remplacé par sa version triée !

```
var fruits = ["Pomme", "Orange", "Banane", "Mangue"];
fruits.sort();
console.log(fruits);           // => ["Banane", "Mangue", "Orange", "Pomme"]
```

⚠ `sort()` considère que les éléments sont des chaînes de caractères :

```
var nombres = [40, 100, 1, 5, 25, 10];
nombres.sort();                // => [1, 10, 100, 25, 40, 5]
```

💡 On peut fournir à `sort()` une *fonction de comparaison* :

```
var nombres = [40, 100, 1, 5, 25, 10];
nombres.sort(function(a, b){return a - b});    // => [1, 5, 10, 25, 40, 100]
```

- **map()** crée un nouveau tableau en **appliquant une fonction à chaque élément** d'un tableau existant :

```
var numbers1 = [45, 4, 9, 16, 25];
```

```
var numbers2 = numbers1.map(function(value) {return value * 2});
```

- **filter()** crée un nouveau tableau avec les éléments d'un tableau existant **qui réussissent un test** :

```
var numbers1 = [45, 4, 9, 16, 25];
```

```
var numbers2 = numbers1.filter(function(value) {return value > 18});
```

- **reduce()** applique une fonction à chaque élément d'un tableau pour **produire une seule valeur** :

```
var numbers1 = [45, 4, 9, 16, 25];
```

```
var numbers2 = numbers1.reduce(function(total, value) {return total + value});
```


Boucles

```
var cars = ["Renault", "Volvo", "BMW"];
```

Boucle **for** :

```
var texte = "";
for (var i = 0; i < cars.length; i++) {
    texte += cars[i] + " ";
}
```

> *texte* contient "Renault Volvo BMW "

Boucle **while** :

```
var i = 0, texte = "";
while (i < cars.length) {
    texte += cars[i] + " ";
    i++;
}
```

> *texte* contient "Renault Volvo BMW "

Boucles

- Enfin, la méthode `Array.forEach()` permet d'appliquer une fonction sur chaque élément d'un tableau :

```
var fruits = ["Banane", "Orange", "Pomme"];
fruits.forEach(myFunction);
```

```
function myFunction(value) {
    console.log(value);
}
```

```
> Banane
  Orange
  Pomme
```

💡 `myFunction()` est une *callback* qui prend jusqu'à 3 arguments : `value`, `index`, `array`

Fonctions fléchées (ES 6)

ES 6 a introduit une nouvelle syntaxe pour écrire des fonctions :

- Avant ES 6 :

```
hello = function() {  
    return "Hello World!";  
}
```

- Avec ES 6 :

```
hello = () => {  
    return "Hello World!";  
}
```

- Et même (si l'unique instruction de la fonction est un *return*) :

```
hello = () => "Hello World!";
```




JavaScript peut manipuler des *objets*, possédant des propriétés et des méthodes :

```
var person = {  
    firstName: "John",  
    lastName : "Doe",  
    age      : 50,  
    eyeColor : "blue",  
    fullName : function() {  
        return this.firstName + " " + this.lastName;  
    }  
};
```

- 💡 Le mot-clé `this` fait référence à l'objet que l'on est en train de manipuler, celui qui possède la propriété / méthode
- 💡 Comparer deux objets renvoie toujours `false`

Il existe un objet particulier : l'« objet » **null** (c'est un objet « sans valeur »).

💡 On l'utilise quand on doit créer un objet, mais qu'on en est dans l'impossibilité :

```
function greetObject(who) {    // renvoie un objet construit à partir de who
  if (!who) {
    return null;
  }
  return { message: 'Hello, ${who}!' };
}

greetObject('World'); // => objet { message: 'Hello, World!' }
greetObject();        // => objet null
greetObject('');     // => objet null aussi !!! (en JS, '' est une falsy value !)
```

⚠ En JS, **null** est bien un type primitif, mais **typeof** renvoie **object** ! (Bug qui remonte aux premières versions de JS)

```
let v = null;
typeof v; => 'object' !  // impossible d'utiliser typeof pour détecter un null !
```



⚠ **null ≠ undefined**

💡 **undefined** désigne une variable qui a été *déclarée* mais pas *initialisée* :

```
let v;  
typeof v; // => "undefined"
```

💡 **null** et **undefined** sont égaux en **valeur** mais pas en **type** :

```
null == undefined // => true  
null === undefined // => false
```

💡 Pour tester si un objet est **null**, on utilise donc l'opérateur de comparaison stricte **==** :

```
const missingObject = null;  
const existingObject = { message: 'Hello!' };
```

```
missingObject === null; // => true  
existingObject === null; // => false
```


Il est possible de créer des classes qui *héritent* d'une autre classe, avec le mot-clé **extends** :

```
class VoitureElectrique extends Voiture {  
    constructor(marque, chargeur) {  
        super(marque);      // appel au constructeur de la classe mère  
        this.typeChargeur = chargeur  
    }  
    afficher() {  
        console.log("Je suis une " + this._marque + " dotée d'un " + this.typeChargeur);  
    }  
}
```

Il est d'usage de manipuler les propriétés d'une classe avec des *accesseurs* (*getters*) et des *mutateurs* (*setters*) :

```
class VoitureElectrique extends Voiture {  
    constructor(marque, chargeur) {  
        super(marque);      // appel au constructeur de la classe mère  
        this.typeChargeur = chargeur  
    }  
    get marque() { return this._marque; }  
    set marque(m) { this._marque = m; }  
}
```

- ⚠ Le nom de l'accesseur / mutateur doit être différent de celui de la propriété correspondante

Tableaux et objets constants



💡 Quand on déclare un tableau ou un objet avec le mot-clé `const`, c'est la *référence* à ce tableau / objet qui est constante

```
const tableau = [1, 2, 3];
tableau[0] = 0;
console.log(tableau);           // affiche [0, 2, 3]
tableau = [0, 2, 3];           // erreur : on ne peut pas réassigner une constante !

const personne = {nom : "Dupont"};
personne.nom = "Durand";        // ok
console.log(personne.nom);     // affiche "Durand"
personne = {nom : "Martin"};   // erreur : on ne peut pas réassigner une constante !
```

Tableaux et objets constants

FREEZE



💡 Pour « figer » le contenu d'un tableau ou d'un objet, on utiliser la méthode `Object.freeze()` :

```
const tableau = [1, 2, 3];
Object.freeze(tableau);
tableau[0] = 0;                                // pas d'erreur signalée
console.log(tableau);                          // affiche [1, 2, 3] : tableau non modifié

const personne = {nom : "Dupont"};
Object.freeze(personne);
personne.nom = "Durand";                      // pas d'erreur signalée
console.log(personne.nom);                    // affiche "Dupont"
```

Evénements

⇒ Se produit lors d'un changement d'état de l'environnement provoqué par :

- **le navigateur** : une page HTML a terminé de charger, une page HTML est fermée...
- **l'utilisateur** : a cliqué sur un bouton, a modifié un champ de formulaire...

Techniquement : objet qui implémente l'interface ***Event***.

Peuvent être surveillés par un ***écouteur d'événements*** (**event listener**)

Peuvent être gérés par un ***manipulateur / gestionnaire d'événements*** (**event handler**)

| | |
|--------------------------|--|
| onchange | un contenu (champ de formulaire, texte...) a été modifié |
| onload | l'élément (page, image...) a été chargé |
| onclick | clic sur un élément |
| ondblclick | double-clic sur un élément |
| onmouseover / onmouseout | la souris passe sur / sort d'un élément |
| onkeydown, onkeyup | appui / relâche d'une touche |
| onkeypress | = onkeydown suivi de onkeyup |
| onfocus / onblur | gain / perte de focus |
| oncopy / oncut | l'utilisateur copie / coupe le contenu d'un élément |

Plus de 85 événements possibles !  https://www.w3schools.com/jsref/dom_obj_event.asp

maPage.html

```
<head>
    <script src='monScript.js'></script>
</head>
<body>
    <form name="myForm" action="/autrepage.html"
        onsubmit="validateForm()" method="post">
        Name: <input type="text" name="fname">
        <input type="submit" value="Submit">
    </form>
</body>
```

monScript.js

```
function validateForm() {
    var x =
        document.forms["myForm"]["fname"].value;
    if (x == "") {
        alert("Name must be filled out");
        return false;
    }
}
```

⇒ Ne pas oublier les options de validation automatiques existant en HTML (**required...**)



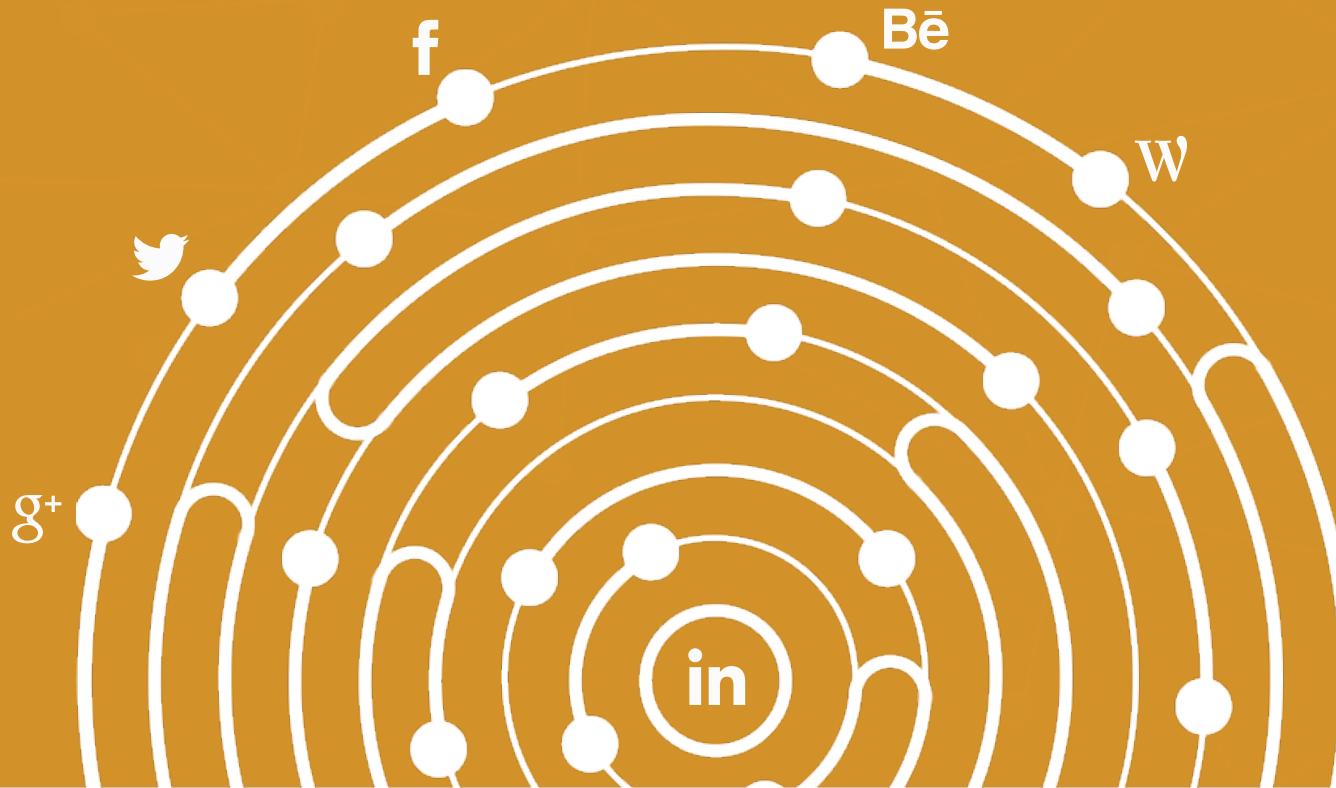
- L'instruction **try...catch** permet de **tester un bloc** de code et de **gérer une éventuelle erreur**
- L'instruction **throw** permet de créer des **erreurs personnalisées**

Exemple : tester si la valeur entrée dans un champ de formulaire est ≥ 5 et ≤ 10 (**test**) :

```
let x = document.getElementById("formInput").value;
try {
    if(x == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
}
catch(err) {
    alert("Input is " + err);
}
```

Bonnes pratiques

- Déclarez toujours vos variables (ou utiliser "use strict";)
- Placez les declarations de variables en haut de vos scripts
- Evitez les variables globales
- Initialisez vos variables
- Attention aux conversions de type automatiques !
- Pensez à l'opérateur === (vérification de la valeur ET du type)
- Terminez chaque instruction par un point-virgule

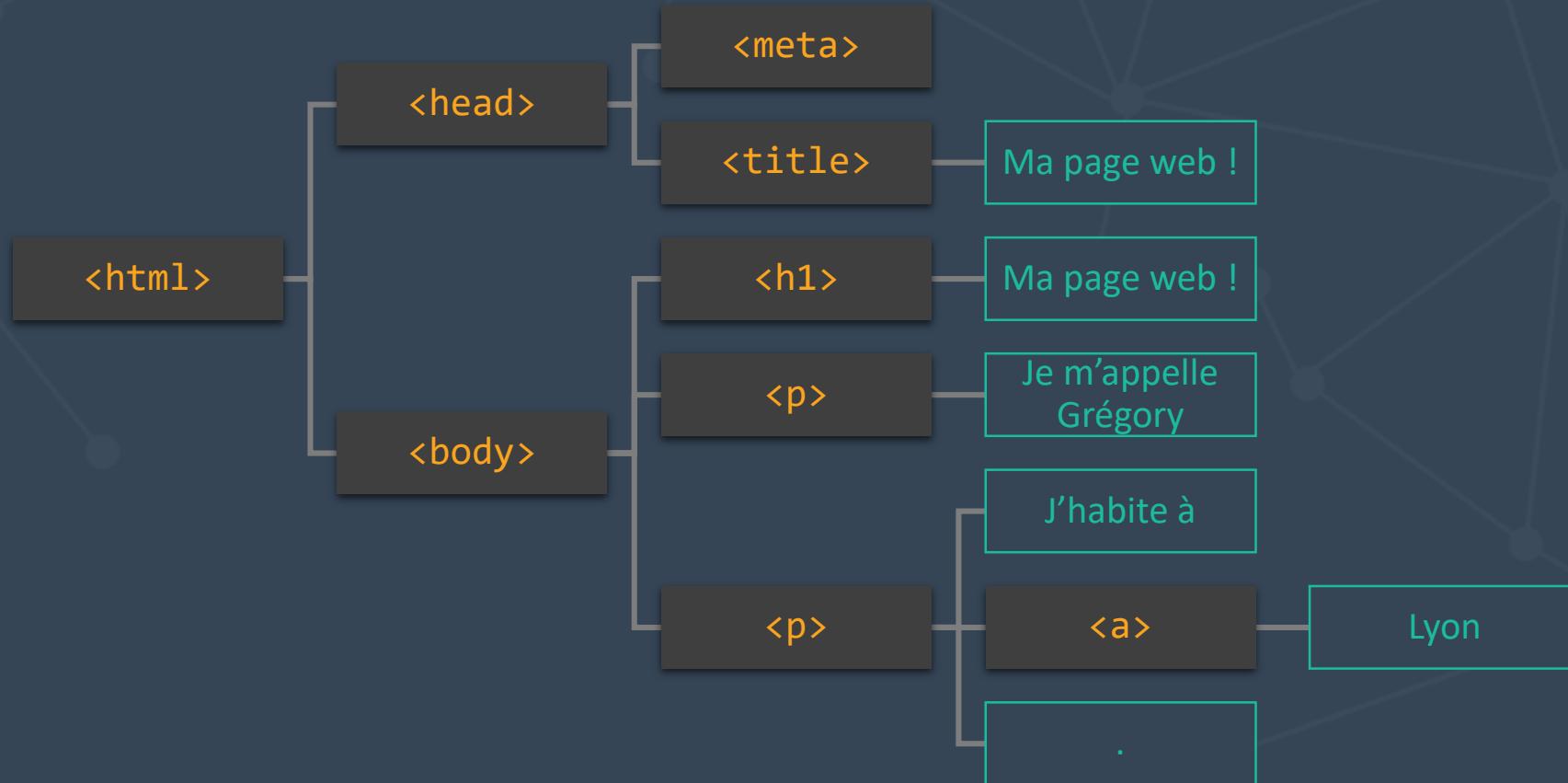


4^{ème} partie : Manipulation du DOM

DOM : Document Object Model

STRUCTURE HIERARCHIQUE

Quand le navigateur charge une page web, il construit son DOM, une **structure arborescente** de tous les objets de la page :



DOM : Document Object Model

UN STANDARD ET UNE API

Le DOM HTML est un **standard** (W3C) et une API HTML (accessible via JavaScript... et d'autres langages)

Avec le DOM, JavaScript peut faire quasiment tout avec la page :

- modifier les **éléments** HTML
- modifier les **attributs** HTML
- modifier les **styles** CSS
- ajouter / supprimer des éléments / attributs HTML
- réagir aux **événements** de la page
- créer de nouveaux événements
- ...

DOM : Document Object Model

ORIENTÉ OBJET

DOM est un modèle orienté **objet** :

- méthodes = **actions** sur les éléments HTML (ajouter, supprimer...)
- propriétés = **valeurs** des éléments HTML (contenu d'un élément HTML...)

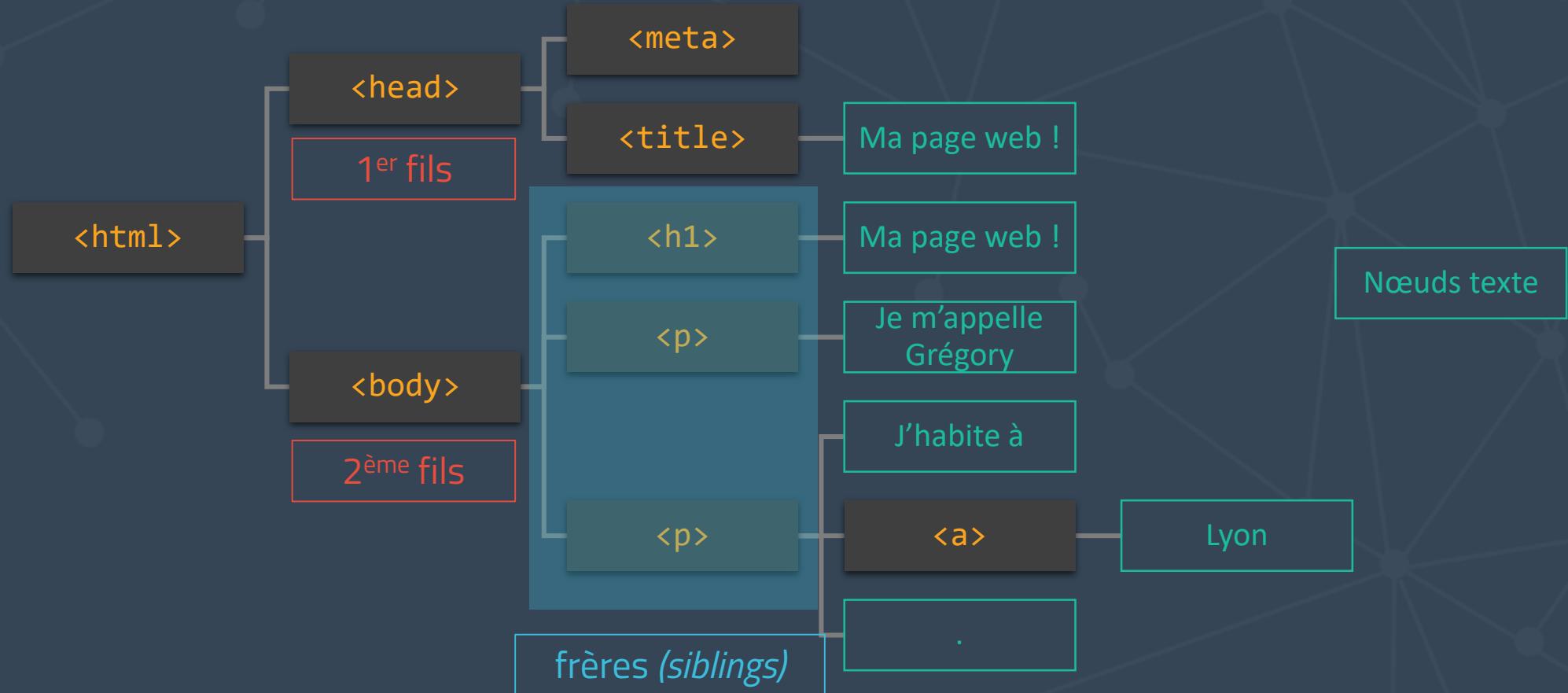
Exemple :

```
<body>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML = "Hello World!";
  </script>
</body>
```

getElementById est une **méthode** ; **innerHTML** est une **propriété**

DOM : Document Object Model

RELATIONS ENTRE LES NOEUDS



DOM : Document Object Model

STRUCTURE HIERARCHIQUE

Naviguer dans la hiérarchie DOM et trouver des éléments HTML :

- `document` : correspond au document *entier*
- `document.documentElement` ($\Leftrightarrow <\text{html}>$)
- `document.head` ($\Leftrightarrow <\text{head}>$) `document.body` ($\Leftrightarrow <\text{body}>$) `document.title` ($\Leftrightarrow <\text{title}>$)
- `parentNode`
- `childNodes[nodenumber]`
- `firstChild`, `lastChild`
- `nextSibling`, `previousSibling`
- `getElementById(id)`, `getElementsByClassName(name)`, `getElementsByTagName(name)`
- `document.forms` : renvoie tous les formulaires, `document.images` : renvoie tous les images
- `document.querySelectorAll(selecteur)` : renvoie les éléments qui correspondent au sélecteur CSS spécifié
- ...

DOM : Document Object Model

L'OBJET `HTMLCollection`

- La méthode `getElementById` renvoie *un* élément
- Mais d'autres méthodes ou propriétés renvoient un *ensemble* d'éléments :
 - `getElementsByName()`
 - `getElementsByClassName()`
 - `forms`
 - `images`

Objet `HTMLCollection` : comme un tableau... mais ce n'est pas un tableau :

- on peut utiliser `length` pour avoir le nombre d'éléments
- on ne peut pas utiliser les méthodes `pop()`, `push()`, `join()`...

DOM : Document Object Model

L'OBJET NodeList

Et d'autres propriétés (`childNodes`) ou méthodes (`querySelectorAll()`) renvoient un objet `NodeList`

💡 Un objet `NodeList` et un objet `HTMLCollection` sont très proches :

- les deux se comportent comme des tableaux d'objets
 - les deux ont une propriété `length`
 - les deux disposent d'un mode d'accès par indice : `myCollection[3]` par exemple
- Mais :
 - on peut aussi accéder aux éléments d'une `HTMLCollection` par leur `nom` ou leur `id`
 - seule une `NodeList` peut contenir des nœuds texte

DOM : Document Object Model

MANIPULER DES ELEMENTS HTML

Les textes sont contenus dans des **nœuds texte** ; on accède à leur valeur par

- `innerHTML / innerText`
- `firstChild.nodeValue / childNodes[0].nodeValue`

Changer des éléments :

- modifier un élément HTML : `element.innerHTML = nouveau contenu HTML`
- modifier un attribut : `element.attribut = nouvelle valeur` OU `element.setAttribute(attribut, valeur)`
- modifier un style : `element.style.propriété = nouveau style`

Ajouter / supprimer des éléments :

`createElement(element), removeElement(element), appendChild(element), replaceChild(new, old)`

Ajouter un gestionnaire d'événement (par exemple réagir à un clic) :

`getElementById(id).onclick = function() {code de la fonction}`

DOM : Document Object Model

MANIPULER DES ELEMENTS HTML

Exemples :

```
var x = document.getElementById("main");
var y = x.getElementsByTagName("p"); // Tous les éléments <p> dans l'élément main

var x = document.getElementsByClassName("intro"); // Tous les éléments de la classe intro

var x = document.querySelectorAll("p.intro"); // Sélection par sélecteur CSS

var x = document.forms["frm1"]; // Sélection du formulaire frm1

document.write(Date()); // Ecrire du code HTML

// Modifier le style :
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
document.getElementById("p2").style.fontSize = "larger";
```

DOM : Document Object Model

MANIPULER DES ELEMENTS HTML

Exemples :

```
// Ajouter un gestionnaire d'événement à un bouton  
document.getElementById("myBtn").onclick = displayDate;
```

```
// ou  
document.getElementById("myBtn").addEventListener("click", displayDate);
```

La différence, c'est qu'avec `addEventListener` on peut ajouter plusieurs gestionnaires au même événement :

```
element.addEventListener("click", myFunction);  
element.addEventListener("click", mySecondFunction);
```

et on peut supprimer un événement avec `removeEventListener` ([exemple](#))

Quelques événements :

- `onload / onunload` : quand la page est chargée / fermée
- `onchange` : quand un élément est modifié (souvent utilisé avec la validation de formulaire)
- `onmouseover / onmouseout` : quand la souris arrive / part d'un élément
- `onmousedown / onmouseup / onclick` : appuis sur les boutons de la souris

DOM : Document Object Model



BOUILLONNEMENT VS CAPTURE



Quiz : on a un `<p>` dans un `<div>`, chacun avec un gestionnaire d'événement réagissant au clic.

On clique sur le paragraphe : quel événement est appelé en premier (celui de `p` ou de `div`) ?

Ça dépend !

Ça dépend de la **propagation** utilisée :

- **bouillonnement (bubbling)** : l'élément le plus **interne** est traité en premier
- **capture** : l'élément le plus **externe** est traité en premier

Spécifié en tant que 3^{ème} paramètre booléen de `addEventListener` :

- `false` = bouillonnement (valeur par défaut)
- `true` = capture

Exemple

JavaScript / DOM offrent la possibilité de **minuter** des événements :

- `setTimeout(function, milliseconds)` : exécute *function* après le délai *milliseconds*
- `setInterval(function, milliseconds)` : id. mais recommence indéfiniment

Pour empêcher l'exécution d'une fonction avant l'expiration du timeout, ou arrêter une boucle :

- `clearTimeout(variable)`
- `clearInterval(variable)`

Exemples :

https://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_animate_3

https://www.w3schools.com/js/tryit.asp?filename=tryjs_setinterval3

Balise <template>

- La balise <template> permet de stocker du contenu HTML (côté client) qui ne doit **pas être affiché** lors du **chargement** de la page mais qui peut être instancié et affiché par la suite grâce à un script JavaScript.
- Ce contenu peut servir de *modèle* et être cloné puis affiché, en JavaScript
- Exemple : on a une liste d'objets *voitures* qu'on souhaite afficher dans une page HTML :

```
let voitures = [  
    { marque: "Ferrari", couleur: "rouge" },  
    { marque: "Porsche", couleur: "noire" }  
];
```

Balise <template>

- Le code HTML correspondant ressemblerait à :

```
<html>  
  <body>  
    <p>Liste des voitures :</p>  
    <template id="listeVoitures">  
      <span>Voici une {{modèle}} de couleur {{couleur}}<br></span>  
    </template>  
  </body>  
  <script src="template.js"></script>  
</html>
```

Syntaxe « moustache »
pour mettre en évidence le
texte qui sera remplacé
dynamiquement

Script inséré APRES
<body> car le template doit
avoir été créé avant de
l'exploiter via JavaScript

Balise <template>

enfin, le code JavaScript :

```
let template = document.querySelector("#listeVoitures");

for (const v of voitures) {                                // itère sur le tableau

    let clone = document.importNode(template.content, true); // clone le template

    newContent = clone.firstElementChild.innerHTML           // remplace {{modèle}}
        .replace(/{{modèle}}/g, v.marque)                   // et {{couleur}} par
        .replace(/{{couleur}}/g, v.couleur);                // leur valeur

    clone.firstElementChild.innerHTML = newContent;

    document.body.appendChild(clone);                      // On ajoute le clone créé

}
```

BOM : Browser Object Model

- Permet à JavaScript de récupérer des **informations sur le navigateur** :
 - sur le *viewport* dans lequel s'affiche le document HTML
 - sur l'écran
 - sur la localisation du document HTML lui-même
 - sur l'historique de navigation
 - permet de gérer des *cookies*
- Ensemble de méthodes et propriétés plus ou moins similaires présentes dans les navigateurs modernes
Mais aucun standard officiel !

BOM : Browser Object Model

OBJET WINDOW

L'objet **window** est supporté par tous les navigateurs

C'est l'élément « racine » : tout appartient à **window**, même le DOM HTML :

- `document.getElementById("header");` est équivalent à
- `window.document.getElementById("header");`

On peut récupérer les dimensions de la fenêtre (**sans les barres d'outils ou de défilement**) avec :

- `window.innerWidth`
- `window.innerHeight`

On peut également **ouvrir, fermer, redimensionner, déplacer** une fenêtre

BOM : Browser Object Model

OBJET SCREEN

L'objet `window.screen` donne des informations sur l'écran de l'utilisateur :

- `screen.width / screen.height`
- `screen.availWidth / screen.availHeight` : idem mais sans la barre des tâches
- `screen.colorDepth` : nb de bits pour représenter une couleur (24 ou 32)

Exemple :

```
<script>  
    alert("Les dimensions de votre écran sont :" + screen.width + " x " +  
          screen.height);  
</script>
```

BOM : Browser Object Model

OBJET WINDOW LOCATION

L'objet `window.location` est utilisé pour récupérer l'URL courante et rediriger vers une nouvelle page

On peut extraire de nombreuses informations :

- l'URL entière : `window.location.href`
- le nom de l'hôte (machine qui héberge la page) : `window.location.hostname`
- le chemin de la page sur l'hôte : `window.location.pathname`
- le protocole utilisé (http, https...) : `window.location.protocol`
- le numéro de port (http = 80 ; https = 443) : `window.location.port` // généralement pas affiché

Rediriger vers une nouvelle page :

- `window.location.assign(URL)`

BOM : Browser Object Model

OBJET WINDOW BROWSER

- On peut récupérer certaines informations sur le navigateur avec l'objet **browser** (Attention ! Pas toujours fiable)
 - `navigator.appName`, `navigator.appVersion`
 - `navigator.platform`, `navigator.language`

Exemple :

```
<script>  
    alert("Votre OS est " + navigator.platform + " et la langue du navigateur est " +  
          navigator.language);  
</script>
```

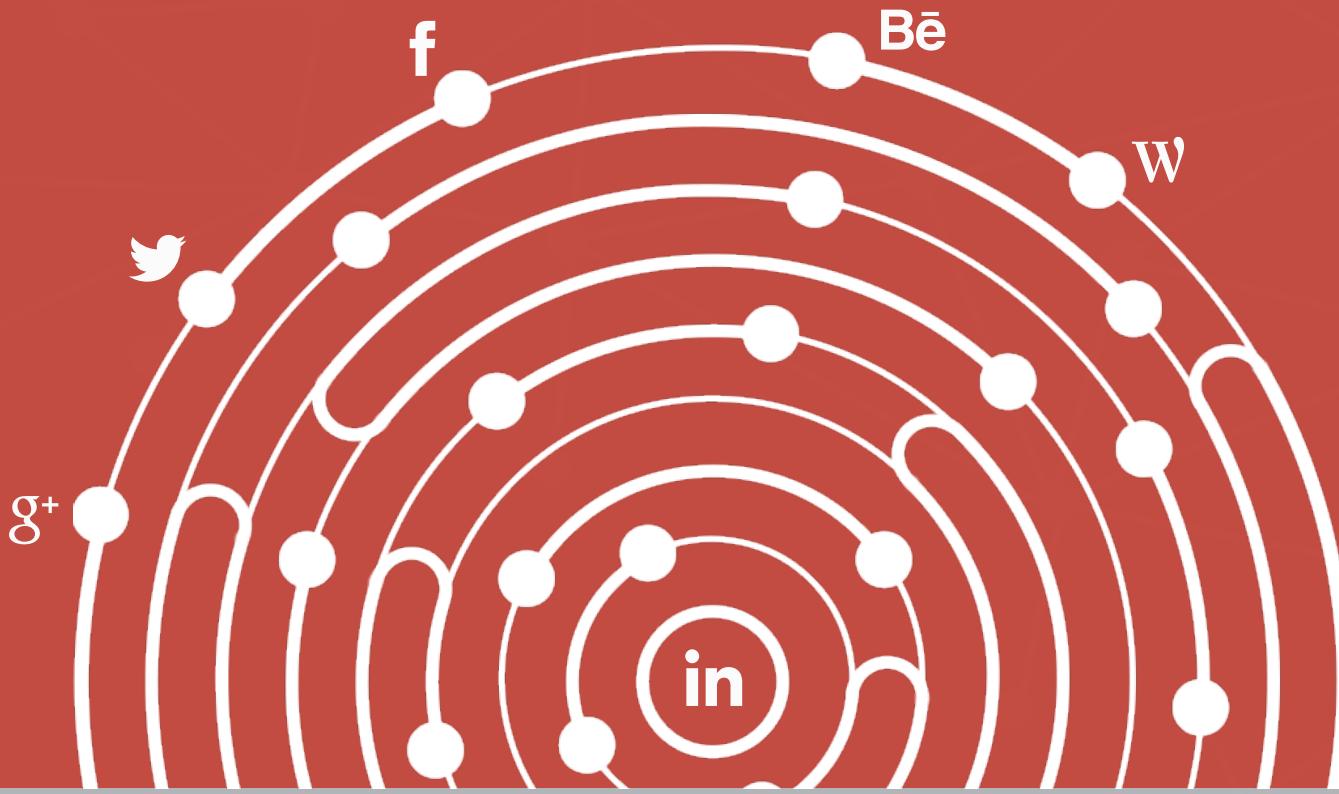
BOM : Browser Object Model

BOITES DE DIALOGUE

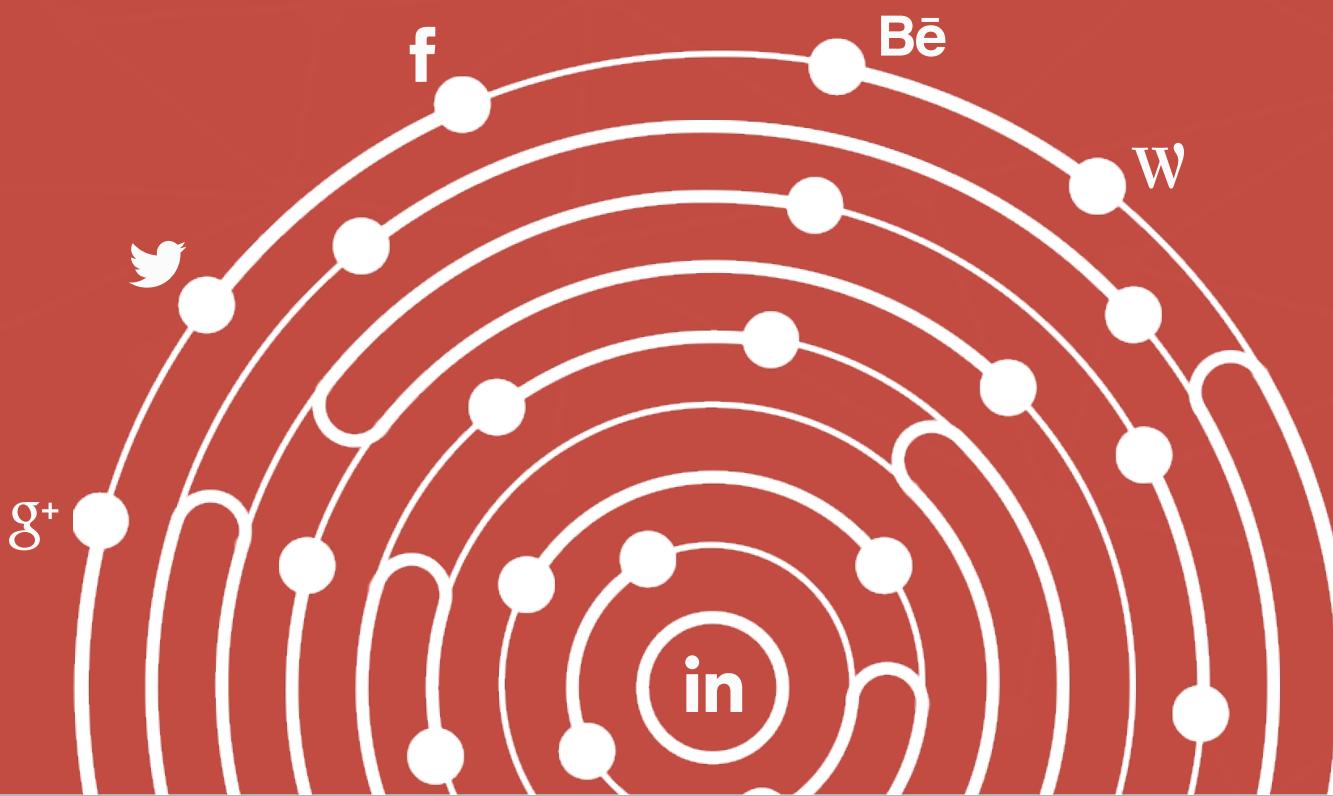
- Boîtes de dialogue : 3 types :
 - **alert** : affichage d'un message ; seulement un bouton *OK*
 - **confirm** : demande de validation ; un bouton *OK* et un bouton *Annuler*
 - **prompt** : demande de saisie d'une valeur ; peut contenir une valeur par défaut

Exemples :

```
alert("Je suis une alerte !");  
confirm("Appuyez sur OK ou Annuler");  
prompt("Entrez votre nom", "Steve Jobs");
```



5^{ème} partie : JavaScript avancé



5.1 : JSON

- Conçu pour stocker et échanger des données
- Fichier texte, écrit avec la **notation JavaScript** utilisée pour les objets :

```
JavaScript : var myObj = {name: "John", age: 31, city: "New York"};
```

JSON :

```
{  
    "name" : "John",  
    "age" : 31,  
    "city" : "New York"  
}
```

- Peut être **directement converti en JavaScript et inversement** :

```
var obj = JSON.parse('{ "name": "John", "age": 30, "city": "New York"}');
```

```
var myJSON = JSON.stringify(obj);
```

JSON vs XML

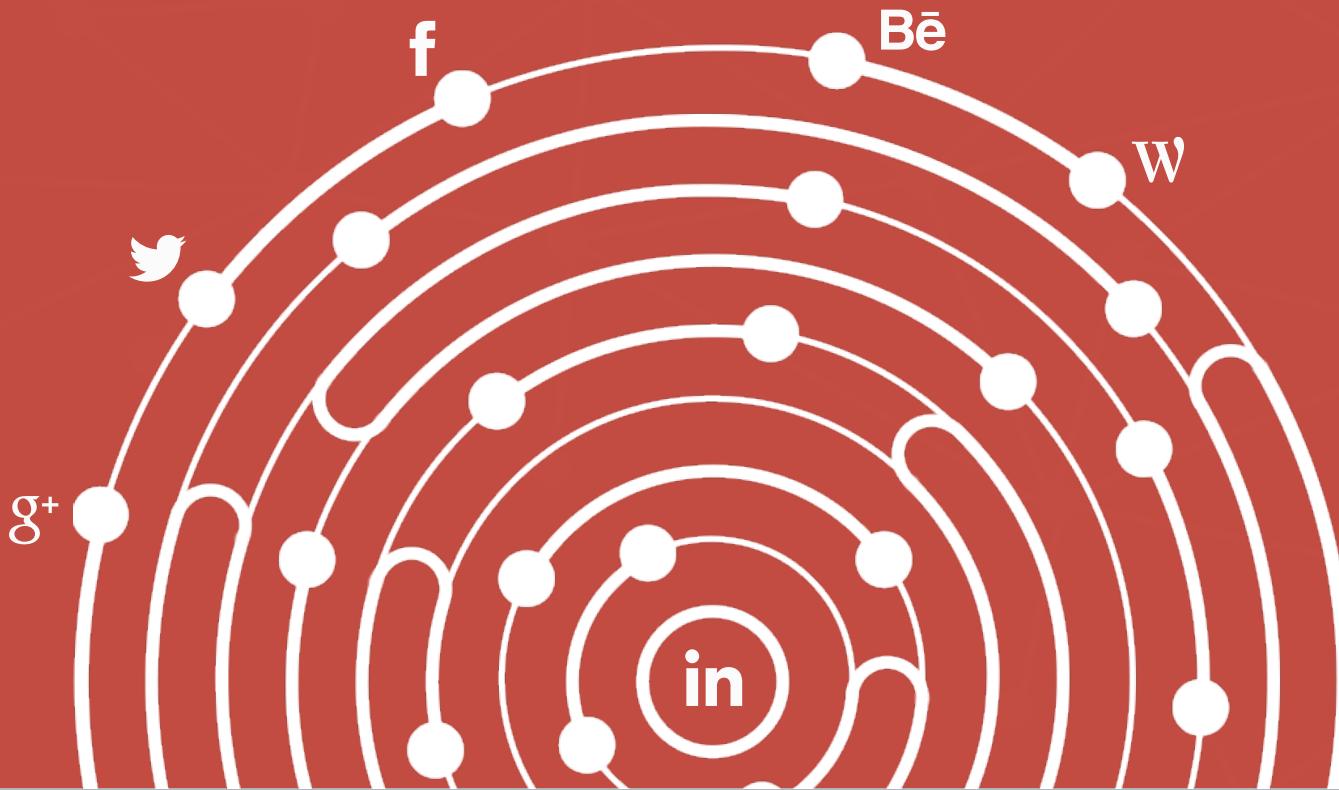
JSON

```
{  
  "person": {  
    "name": "John",  
    "age" : 31,  
    "city": "New York"  
  }  
}
```

XML

```
<person>  
  <name>John</name>  
  <age>31</age>  
  <city>New York</city>  
</person>
```

JSON est plus simple à parser, plus facile à lire par un humain, peut utiliser des tableaux



5.2 : Programmation asynchrone, Promesses, Fetch



Programmation synchrone vs asynchrone

- Code synchrone :

```
function calculerMoyenne() {  
    let notes = recupererNotesSurServeur();  
    let moyenne = somme(notes) / nombre(notes);  
    return notes  
};
```

- *Synchrone* : chaque appel de fonction met `calculerMoyenne` en pause jusqu'à ce que la fonction appelée se termine (appel de fonction *bloquant*)

Si l'appel de fonction met beaucoup de temps, le site semble *bloqué / figé*

(ex. : <https://mdn.github.io/learning-area/javascript/asynchronous/introducing/simple-sync-ui-blocking.html>)

💡 C'est parce qu'on n'utilise qu'un seul *thread*



Programmation synchrone vs asynchrone

- (pseudo)-Code asynchrone :

```
function voyageurCommerce() {  
    r = calculerRouteOptimale(20);      //plus court chemin passant une seule fois par 20 villes
```

Quand r sera disponible :

```
    afficher(r);
```

Pendant ce temps :

Laisser la main à l'utilisateur

```
}
```

- La fonction *calculerRouteOptimale* est exécutée « en parallèle » de *voyageurCommerce*
- JavaScript fournit un moyen simple d'écrire du code asynchrone : les *promesses*



- Une *Promesse* est un objet (instance de la classe *Promise*) qui représente une opération à exécuter avec la *promesse* qu'un résultat sera retourné à moment donné (sans savoir quand précisément)
- Une promesse comporte donc un processus qui va continuer à se dérouler **en parallèle** du reste de notre environnement
- Elle se termine forcément :
 - soit en **succès** (elle est *résolue*) : une fonction s'est terminée, une ressource a été trouvée, etc.
 - soit en **échec** (elle est dite *rejetée*) : une erreur a eu lieu dans la fonction appelée, une ressource est introuvable, etc.

Le constructeur de la classe **Promise** attend en argument une fonction, qui prend elle-même en argument 2 autres fonctions (qui sont elles-mêmes des promesses) :

- **resolve** : indique un succès
- **reject** : indique un d'échec

Exemple :

```
const p = new Promise((resolve, reject) => {
    si succès :
        resolve(...);
    si échec :
        reject(...);
});
```

Pour exploiter le résultat d'une promesse, un objet **Promise** dispose :

- d'une méthode **then()** : c'est le code qui sera exécuté dès que la promesse sera tenue
- d'une méthode **catch()** : c'est le code qui sera exécuté si la promesse est rompue

Promesses

EXEMPLE CONCRET

```
function f() {  
    return new Promise((resolve, reject) => {  
        let nb = Math.round(Math.random() * 20);  
        if(nb % 2 == 0)  
            resolve(nb + " est un nombre pair !");  
        else  
            reject("Oh non, on a tiré un nombre impair : " + nb);  
    });  
  
f()  
.then(nb => console.log(nb))  
.catch(err => console.log(err))
```

💡 Le plus souvent, on utilise des promesses existantes, comme celle de l'API Fetch



- Alternative moderne à AJAX / XMLHttpRequest
- Apparue vers 2015 avec ES6, repose sur la notion de Promesse

Exemple :

```
const myImage = document.querySelector('img');

fetch('flowers.jpg')           // chemin de la ressource à récupérer
.then(function(response) {     // en cas de succès, on obtient une réponse HTTP
    return response.blob();    // (pas directement l'image) ; blob() extrait les données
})
.then(function(myBlob) {        // quand la fonction blob() a terminé...
    const objectURL = URL.createObjectURL(myBlob); // ...on extrait l'URL de la ressource...
    myImage.src = objectURL; // ...et on modifie notre élément img
});
```

⚠ Une promesse Fetch peut être tenue sans avoir réussi à récupérer la ressource demandée (par exemple, une erreur 404 n'empêche pas la promesse d'être tenue)

Il faudrait donc toujours aussi tester la réponse :

```
fetch('flowers.jpg').then(function(response) {
  if(response.ok) {
    response.blob().then(function(myBlob) {
      var objectURL = URL.createObjectURL(myBlob);
      myImage.src = objectURL;
    });
  } else {
    console.log('Mauvaise réponse du réseau');
  }
}).catch(function(error) {
  console.log('Il y a eu un problème avec l\'opération fetch: ' + error.message);
});
```

Exemple d'utilisation de *fetch* pour récupérer le contenu d'un fichier JSON :

```
let contenu_json = {};  
  
fetch('fichier.json')  
.then(function(response) {  
    return response.json();  
})  
    // on spécifie le chemin de la ressource à récupérer  
    // renvoie une Promesse contenant un objet de type Response  
    // c'est une réponse HTTP -> json() extrait la partie JSON  
    // et la convertit en objet JavaScript  
  
.then(function(json) {  
    console.log(json["person"]["age"]);  
    // ou  
    console.log(json.person.age);  
  
    contenu_json = json;  
});  
    // cet objet est récupéré ici (nommé ici json)  
    // -> on peut donc lire facilement son contenu  
    // et en conserver une copie disponible en dehors  
    // de la promesse
```

Attention ! Croisement de domaines

Pour des raisons de sécurité, les navigateurs interdisent aux scripts d'accéder à des données sur des **domaines différents** (*same origin policy*)

⇒ Par défaut, il est impossible depuis un script situé sur <http://monsite.com> d'accéder à des ressources situées sur <http://unautresite.com>, ni même sur <http://m.monsite.com> ou <https://monsite.com> !

💡 Pour pallier cette restriction souvent trop contraignante, un mécanisme appelé *politiques CORS* (Cross-Origin Resource Sharing) a été créé

⚠ Pour des raisons de sécurité, les politiques CORS empêchent aux navigateurs l'accès à des données locales
⇒ dans ce cas, il faut absolument passer par un *serveur web, même en local*

Async / Await

SUCRE SYNTAXIQUE POUR LES FONCTIONS ASYNCHRONES

- Rendent le code asynchrone plus facile à lire et écrire
- Ajoutés à JavaScript dans **ES8 (2017)**
- **async** : transforme une fonction en fonction *asynchrone*, qui retourne une promesse

```
async function hello() { return "Bonjour" };  
hello();           // <- retourne désormais une promesse
```

- **await** : met en pause le code jusqu'à ce que la promesse soit réalisée

=> Donne une apparence synchrone au code

Async / Await

SUCRE SYNTAXIQUE POUR LES FONCTIONS ASYNCHRONES

Exemple :

Sans async / await

```
function testFetch() {  
  fetch('fichier.json')  
    .then(function(reponse) {  
      return reponse.json();  
    })  
    .then(function(json) {  
      console.log(json.person.age);  
    });  
}
```

Avec async / await

```
async function testFetch() {  
  const reponse = await fetch('fichier.json');  
  const json = await reponse.json();  
  console.log(json.person.age);  
  
  return json;  
}
```

AJAX : Asynchronous JavaScript And XML

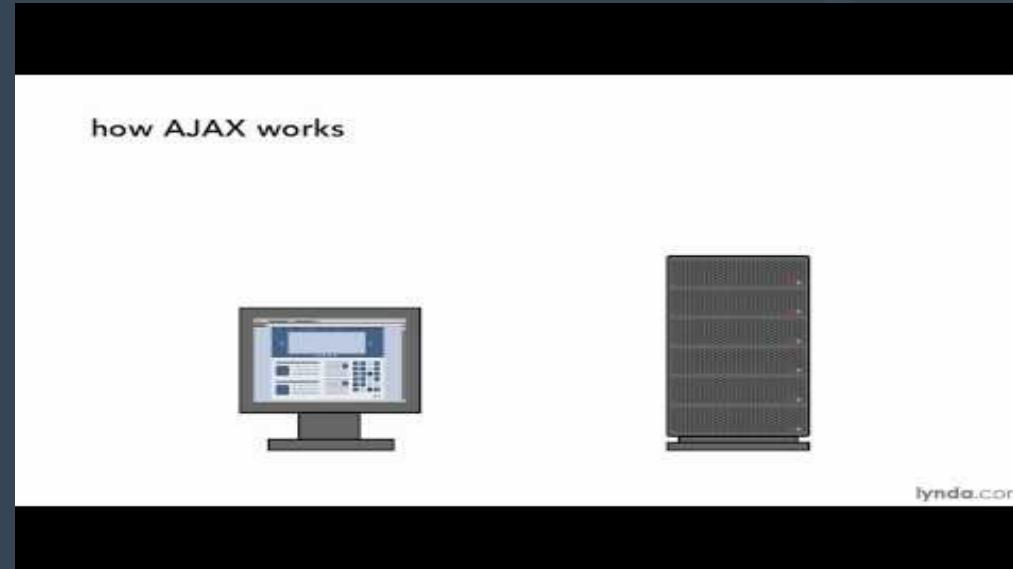


QU'EST-CE-QUE C'EST ?

AJAX n'est *pas* un énième langage à découvrir

C'est un *ensemble de technologies* (JavaScript, DOM, XML, JSON...) qui, mises ensemble, permettent de :

- Récupérer des données sur un serveur **après que la page a été chargée**
- Actualiser une page **sans la recharger**
- Envoyer des données à un serveur **(en arrière-plan)**



AJAX : Asynchronous JavaScript And XML

COMMENT CA MARCHE ?



Client

- Un événement intervient...
- un objet *XMLHttpRequest* est créé
 - la requête HTTP est envoyée

Internet

Serveur

- traite la requête HTTP
- crée une réponse et la renvoie au client

- traite la réponse reçue en utilisant JavaScript
- le contenu de la page est mis à jour

Internet

1. Créer un objet XMLHttpRequest :

```
var xhttp = new XMLHttpRequest();
```

2. Demander une ressource (fichier texte ou XML, résultat d'un script, etc.) :

```
xhttp.open("GET", "ajax_info.txt", true);      // true = asynchrone (le laisser à true)  
xhttp.send();                                // donner une chaîne si on est en POST
```

GET ou POST ?

- GET est plus simple et plus rapide ; quand on veut simplement récupérer des informations
- Utiliser POST quand :
 - on veut mettre à jour un fichier ou une base de données
 - on veut envoyer beaucoup de données (POST n'est pas limité en taille)
 - envoyer des données saisies par l'utilisateur (plus sécurisé, et plus robuste aux encodages)
 - on veut être sûr d'avoir la dernière version de la ressource (càd pas mise en cache par le navigateur)

La propriété **onreadystatechange** permet de définir la fonction à exécuter à réception de la réponse :

```
xhttp.onreadystatechange = function () {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML = this.responseText;
  }
};
```

- **readyState** : 1 = connexion établie, 3 = requête en cours de traitement ; 4 = réponse prête
- **status** : 200 = "OK" ; 403 = "Forbidden" ; 404 = "Page not found"
- réponse soit sous forme de texte (**responseText**) soit sous forme XML (**responseXML**)

onreadystatechange est appelée à chaque fois que **readyState** change (donc au moins 4 fois)



Exemple complet :

```
<button type="button" onclick="loadDoc()">Modifier le contenu</button>

<script>
function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML =
                this.responseText;
        }
    };
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
}
</script>
```



- XML = eXtensible Markup Language
- Descendant de SGML (cf. premier cours)
- Conçu pour stocker et manipuler des données
- Conçu pour être lisible à la fois par les humains et les ordinateurs

Exemple :

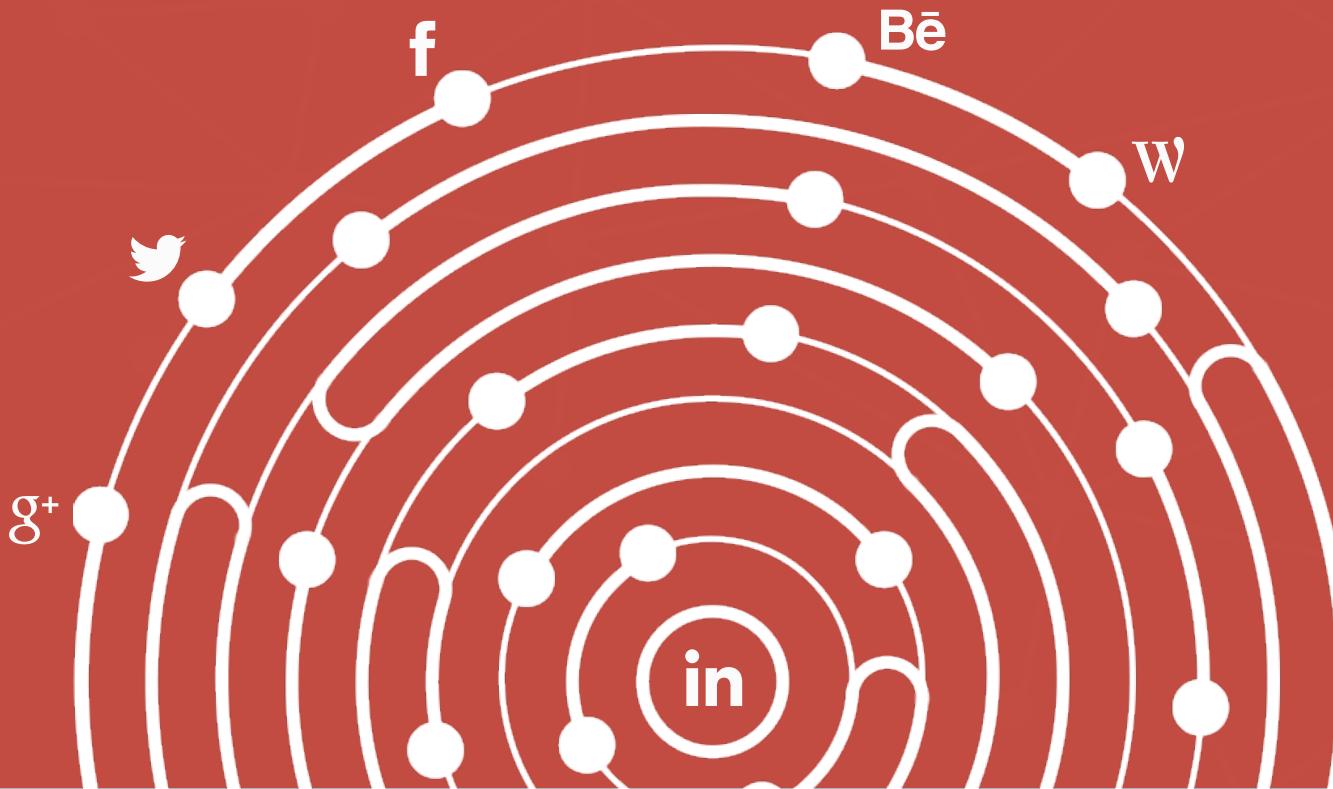
```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  ...

```



Exemple :

```
function loadDoc() {  
    ...  
    xhttp.open("GET", "cd_catalog.xml", true);  
    ...  
}  
  
function myFunction(xhttp) {  
    var xmlDoc = xhttp.responseXML;  
    var table = "<tr><th>Title</th><th>Artist</th></tr>";  
    var x = xmlDoc.getElementsByTagName("CD");  
  
    for (var i = 0; i < x.length; i++) {  
        table += "<tr><td>" +  
            x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue + "</td><td>" +  
            x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue + "</td></tr>";  
    }  
  
    document.getElementById("demo").innerHTML = table;  
}
```



5.3 : Stockage de données

Comment stocker des données ?

COOKIES

Rappel : HTTP est un protocole *sans état (stateless)*

- un serveur web « oublie » les informations de l'utilisateur entre deux connexions
- ⇒ le cookie répond à cette problématique



Cookie : fichier texte contenant quelques données sur l'utilisateur, stocké **sur l'ordinateur de l'utilisateur**



Quand un navigateur demande une page à un serveur, il joint **le cookie à la requête** ; ainsi le serveur se « rappelle » de l'utilisateur

Exemple :

- un utilisateur visite un site ; son nom et son prénom sont enregistrés dans un cookie
- la prochaine fois que l'utilisateur viendra sur ce site, le cookie se « rappellera » de ses noms et prénoms

Comment stocker des données ?

COOKIES

Les cookies contiennent des **chaînes de caractères** qui représentent des données du type **clé = valeur**

La manipulation des cookies en JavaScript se fait via l'objet... **cookie**

```
document.cookie = "username=John Doe";
```

/!\ Par défaut, un cookie est supprimé à la fermeture du navigateur

On peut ajouter une **date d'expiration** :

```
document.cookie = "username=Tim Berners-Lee; expires=Mon, 25 Feb 2019 18:00:00 UTC";
```

La date d'expiration permet de **supprimer un cookie** :

```
document.cookie = "username=Tim Berners-Lee; expires=Thu, 01 Jan 1970 00:00:00 UTC";
```

Comment stocker des données ?

UTILISER LES COOKIES

L'objet `document.cookie` renvoie tous les cookies de la page **dans une seule chaîne**

Il n'existe pas de fonction intégrée permettant de rechercher un cookie particulier !

Un exemple complet :

https://www.w3schools.com/js/tryit.asp?filename=tryjs_cookie_username

(voir https://www.w3schools.com/js/js_cookies.asp pour toutes les explications)



Comment stocker des données ? (suite)

Pour le stockage **côté client**, l'API **Web Storage** apporte une solution moderne aux nombreux problèmes liés aux cookies.

2 interfaces disponibles :

- **sessionStorage** :
 - données stockées le temps d'une session de navigation
 - portée limitée à l'onglet actif
 - stockage limité à un domaine
- **localStorage** :
 - données stockées sans limite de durée
 - possible de l'exploiter à travers plusieurs onglets / fenêtre pour un **même domaine**

💡 Pas d'échange de données entre différents navigateurs

Utilisation (identique pour `sessionStorage` et `localStorage`) :

- Ecriture :

```
sessionStorage.setItem("couleur", "vert"); ou sessionStorage.couleur = "vert";
```

- Lecture :

```
var c = sessionStorage.getItem("couleur"); ou var c = sessionStorage.couleur
```

- Suppression :

```
sessionStorage.removeItem("couleur");
```

- Suppression totale :

```
sessionStorage.clear();
```

 Toutes les données sont stockées sous forme de chaînes de caractères

Et si on souhaite stocker des *objets* ?

- Ecriture :

```
var monobjet = {  
    propriete1 : "valeur1",  
    propriete2 : "valeur2"  
};  
  
var monobjet_json = JSON.stringify(monobjet); // « linéarisation »  
sessionStorage.setItem("objet",monobjet_json);
```

- Lecture

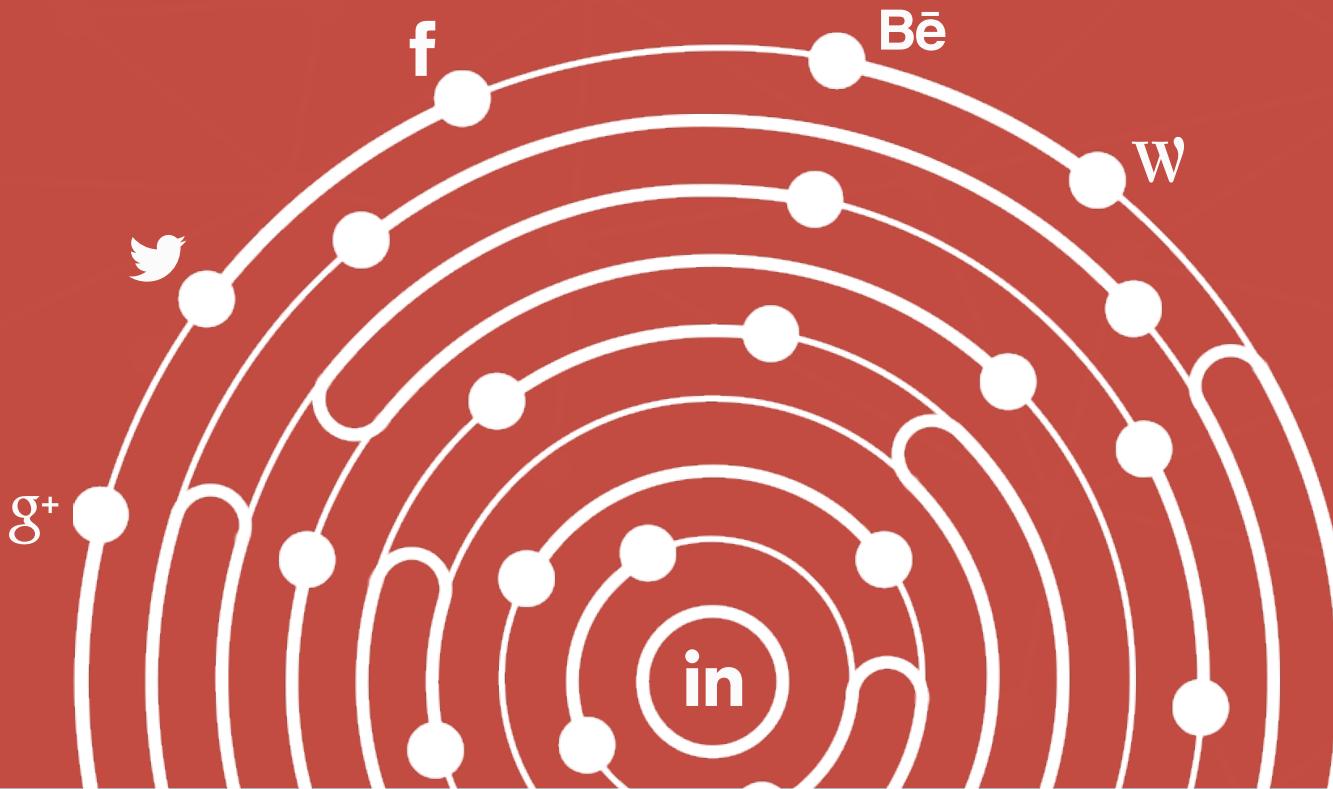
```
var monobjet_json = sessionStorage.getItem("objet");  
var monobjet = JSON.parse(monobjet_json);  
// Affichage dans la console  
console.log(monobjet);
```

Et si on souhaite stocker *beaucoup* de données et / ou des données *binaires* ?

IndexedDB



- 💡 Base de données de type *NoSQL*
- 💡 Traitement *asynchrone* (pas de « freeze » de l'application)
- Mini tuto : https://www.tutorialspoint.com/html5/html5_indexeddb.htm
- 💡 Remplace l'ancienne API (dépréciée) WebSQL



5.4 : jQuery

Qu'est-ce que c'est ? :

- bibliothèque JavaScript créée pour simplifier JavaScript (appels AJAX, manipulation du DOM...)
- créée en 2006 par un étudiant, John Resig
- facile à apprendre
- “Ecrire moins pour faire plus”
- utilisé par toutes les grandes entreprises (GAFAM, IBM...)

Utilisation :

- soit la télécharger depuis le site officiel www.jquery.com
- soit utiliser un CDN (Content Delivery Network) : Google, Microsoft...

```
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
</head>
```

Syntaxe : `$(selector).action()`

- le symbole `$` indique qu'on écrit du jQuery
- un sélecteur (comme en CSS...) pour trouver ("query") un ou des éléments HTML
- une action jQuery à exécuter sur ces éléments

Exemples :

`$(this).hide()` – cache l'élément courant

`$("p").hide()` – cache tous les éléments `<p>`

`$(".test").hide()` – cache tous les éléments de la *classe* "test"

`$("#test").hide()` – cache l'élément ayant *pour id* "test"

Syntaxe :

```
$("p").click(function() {  
    ... code à exécuter lors d'un clic  
})
```

Événements disponibles (extrait) :

| Mouse Events | Keyboard Events | Form Events | Document/Window Events |
|--------------|-----------------|-------------|------------------------|
| click | keypress | submit | load |
| dblclick | keydown | change | resize |
| mouseenter | keyup | focus | scroll |
| mouseleave | | blur | unload |

La fonction `on(...)` permet d'attacher plusieurs gestionnaires d'événements à un objet en une seule fois :

```
$("p").on({  
    mouseenter: function(){  
        $(this).css("background-color", "lightgray");  
    },  
  
    mouseleave: function(){  
        $(this).css("background-color", "lightblue");  
    },  
  
    click: function(){  
        $(this).css("background-color", "yellow");  
    }  
});
```

Récupérer / définir des infos HTML :

- `text()` : contenu textuel des éléments sélectionnés
- `html()` : contenu HTML des éléments sélectionnés
- `val()` : valeur des champs de formulaire

Modifier un attribut :

- `attr(attribut, valeur)`
Ex. : `$("#lien").attr("href", "https://www.cpe.fr");`

Ajouter / Supprimer des éléments HTML :

- `append()`, `prepend()`, `before()`, `after()`
- `remove()`, `empty()`

Bonne pratique : encadrer le code jQuery par

```
$(document).ready(function(){  
    ...  
});
```

ou même encore plus simplement (mais moins clair à la lecture) :

```
$(function(){  
    ...  
});
```

Avantages :

- le code ne sera exécuté que lorsque la page sera complètement chargée
- on peut insérer le script dans la section **head**

jQuery simplifie l'écriture de requêtes AJAX !

```
$(selector).load(URL, data, callback);
```

`URL` = adresse de la ressource à charger

`data` = données (sous forme *clé / valeur*) à envoyer avec la requête

`callback` : fonction à exécuter après réception de la ressource (*optionnel*)

La `callback` peut comporter les paramètres suivants :

- `responseTxt` : contenu de la réponse en cas de succès
- `statusTxt` : chaîne de caractères indiquant le statut de la requête ("success" / "error")
- `xhr` : l'objet XMLHttpRequest

Exemple :

```
$(“button”).click(function () {  
    $(“#div1”).load(“demo_test.txt”, function (responseTxt, statusTxt, xhr) {  
        if (statusTxt == “success”)  
            alert(“External content loaded successfully!”);  
  
        if (statusTxt == “error”)  
            alert(“Error: ” + xhr.status + “: ” + xhr.statusText);  
    });  
});
```

Envoyer des requêtes GET ou POST :

```
$.get(URL,callback);  
$.post(URL,data,callback);
```