

# **HTML & CSS**

---

## **Les langages de base du web**

# HTML

---

*HyperText Markup Language*

# Origine

---

~ 1990, Tim Berners-Lee @ CERN

Invention du World Wide Web = convergence de 3 technologies :

- URL (Universal Resource Locator) : protocole d'adressage de ressources
- HTTP (HyperText Transfert Protocol) : protocole de communication client/serveur
- **HTML (HyperText Markup Language) : langage de description de document**

Langage à balises issu du **SGML** : arbre d'**éléments + attributs**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>A very simple page</title>
  </head>
  <body>
    <p>Hello world! <a href="https://scihub.org">Find great articles here</a></p>
    <!-- This is a comment -->
  </body>
</html>
```

# Principes fondateurs

---

- Format texte ouvert
- Facile à lire (pour les machines et les humains)
- Sémantique, accessible
- Robuste, résilient (pas de "plantage")

Principes parfois (et toujours !) menacés (versions propriétaires, monopoles, etc.)

- 1994 : [W3C](#) créé par Tim-Berners Lee pour protéger et faire évoluer les standards
- ~2000 : Poussée du W3C vers le XML (XHTML)
- 2004 : [WHATWG](#), groupement de développeurs de navigateurs  
Approche plus pragmatique du langage, en lien avec les implémentations
- Depuis : maintenu conjointement par le W3C et le WHATWG

## Version actuelle

---

**HTML** "*Living standard*", ou "HTML5" par abus de langage

- Spécifications du langage
- [MDN \(Mozilla Developer Network\)](#)

Référence incontournable pour le développement

Abandon de l'approche XML stricte :

- certains éléments n'ont pas besoin d'être fermés (ex : `<meta>` , `<br>`)
- certains attributs peuvent être sans valeur (ex : `checked` , `download` )



Actuellement, ~ 115 éléments (voir [HTML Tags Memory Test](#))

## Base

---

- `<!DOCTYPE html>` : déclaration de type de document.
- `html` : racine, unique
  - `head` : en-tête, unique
  - `body` : corps, unique

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <!-- en-tête -->
  </head>
  <body>
    <!-- corps -->
  </body>
</html>
```

## En-tête <head>

- Titre de la page
- Ressources annexes (icônes, feuilles de styles CSS, JavaScript, flux RSS, etc.)
- Métadonnées (pour navigateurs, moteurs de recherche, réseaux sociaux, etc.)

```
<head>
  <meta charset="utf-8" />

  <title>Articles - Blog de Benjamin Becquet</title>

  <meta name="description" content="Liste de mes articles" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />

  <link rel="stylesheet" media="screen" href="style.css" />
  <link rel="alternate" type="application/rss+xml" href="feed.xml" />
  <link rel="icon" href="logo.png" />

  <script src="stats.js"></script>
</head>
```

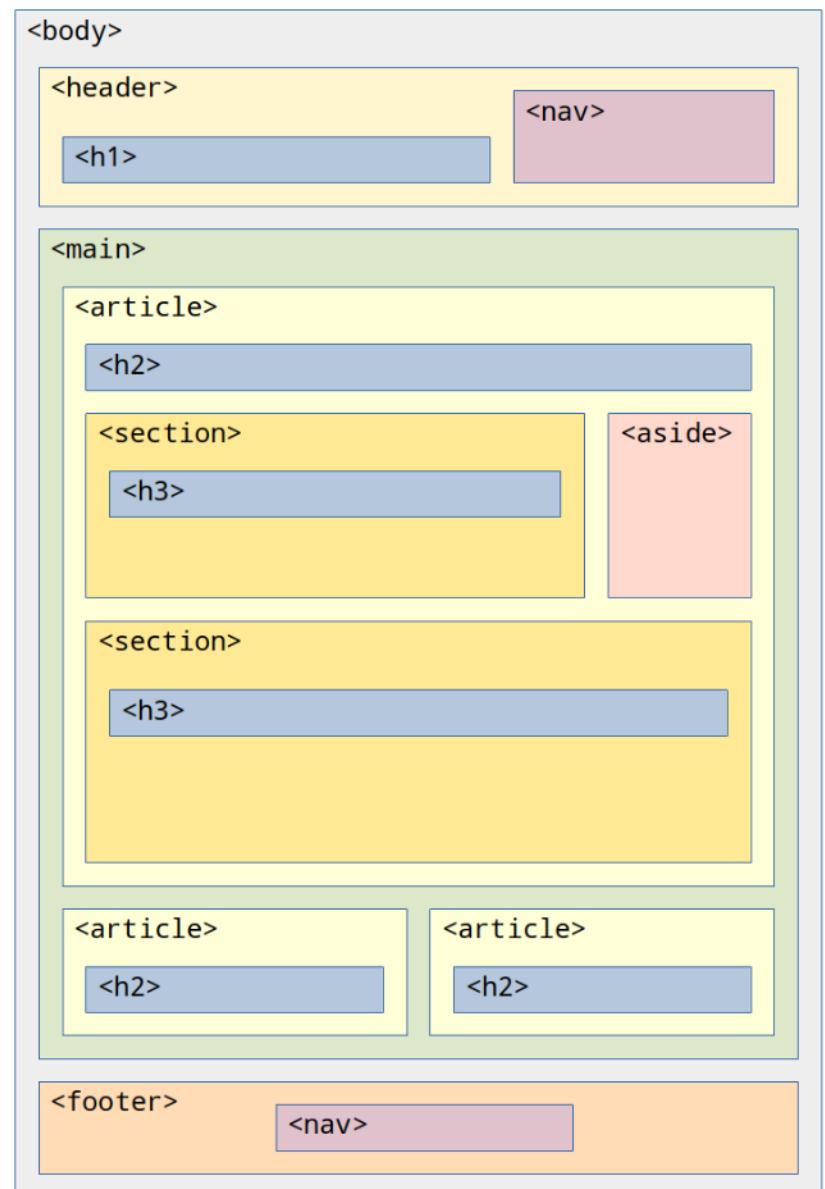
# Structure macro du document

- Blocs sémantiques

```
<main>, <header>, <footer>, <nav>, <article>,  
<section>, <aside>, <figure>
```

- Hiérarchie de titres

```
<h1>, <h2>, ... <h6>
```



# Formatage du contenu

---

- Paragraphes : `<p>`
- Typographie basique : `<b>` , `<i>` , `<em>` , `<sup>` , `<sub>`
- Listes :
  - Numérotées (*ordered*) : `<ol>` + `<li>`
  - Non numérotées (*unordered*) : `<ul>` + `<li>`
  - De définitions : `<dl>` + `<dt>` + `<dd>`
- Citations :
  - en ligne : `<q>`
  - en bloc : `<blockquote>` + `<cite>`
- Blocs de code préformatté : `<pre>` + `<code>`

## Liens

Tous les liens hypertextes se font avec la balise `<a>` (pour *anchor*) et son attribut `href` :

```
<a href="https://wikipedia.org/wiki/HTML">Lien absolu</a>  
  
<a href="../../cours-react.html">Lien relatif</a>  
  
<a href="/articles/2020/10/technoweb">Lien relatif à la racine du site</a>  
  
<a href="#section1">Lien interne (vers l'élément d'id "section1")</a>  
  
<a href="mailto:benjamin.becquet@gmail.com">Lien mail</a>  
  
<a href="tel:0123456789">Lien téléphone</a>
```

Quelques attributs utiles :

- `target="_blank"` ouvrir la cible dans un nouveau "contexte" (onglet) ! À utiliser à bon escient.
- `rel` : pour qualifier la relation entre source et cible ( `nofollow` , `noopener` , `noreferrer` , etc.)
- `download` : pour un lien de téléchargement

# Ressources externes

- Images :

```

```

- Plusieurs versions selon l'affichage (*responsive*) et le support des formats :

```
<picture>
  <source srcset="logo-768.png 768w, logo-768-1.5x.png 1.5x" type="image/webp" />
  <source srcset="logo-480.png, logo-480-2x.png 2x" />
  
</picture>
```

- Contenu multimédia :

`<video>` & `<audio>`, avec mécanisme similaire à `<picture>` pour le support des formats

- Insert d'un autre document HTML :

```
<iframe src="url" title="description"></iframe>
```

# Tableaux de données

```
<table>
  <caption>
    Achats bricolage
  </caption>
  <thead>
    <tr>
      <th>Produit</th>
      <th>Prix</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Pinceaux</td>
      <td>20 €</td>
    </tr>
    <tr>
      <td>Peinture</td>
      <td>55 €</td>
    </tr>
    <tr>
      <td>Marteau</td>
      <td>12 €</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <th>Total</th>
      <td>87 €</td>
    </tr>
  </tfoot>
</table>
```

Achats bricolage

Produit	Prix
Pinceaux	20 €
Peinture	55 €
Marteau	12 €
<b>Total</b>	<b>87 €</b>



Pas pour organiser visuellement du contenu dans une page

## Formulaires <form>

---

Éléments interactifs et *stateful* (maintiennent une valeur interne), base de l'interactivité du web

- <input>, décliné via son attribut `type` (voir [liste complète](#))

- `type="text"`

- `type="checkbox"`

- `type="radio"`

- `type="range"`

- <select> + <option> + <optgroup>

- <textarea>

- + <label>, associé à l'élément via l'attribut `for`

# Formulaires

```
<form action="/createAccount" method="POST">
  <div class="field">
    <label for="mail">Adresse mail</label>
    <input id="mail" name="mail" type="email" />
  </div>
  <div class="field">
    <label for="password">Mot de passe</label>
    <input id="password" name="password" type="password" />
  </div>
  <div class="field">
    <label for="gender">Sexe</label>
    <select id="gender" name="gender">
      <option value="nc">Sans indication</option>
      <option value="m">Homme</option>
      <option value="f">Femme</option>
    </select>
  </div>
  <div class="field">
    <input id="tos" name="tos" type="checkbox" />
    <label for="tos">J'ai lu les conditions d'utilisation</label>
  </div>
  <button type="submit">Créer mon compte</button>
</form>
```

The screenshot shows a web browser window with the title "Technoweb/form.html". Inside the window, there is a form for account creation. The form consists of several input fields and a submit button. The fields are labeled in French: "Adresse mail" (Email address), "Mot de passe" (Password), "Sexe" (Gender), and "J'ai lu les conditions d'utilisation" (I have read the terms of service). The "Sexe" field has a dropdown menu open, showing options "Sans indication", "Homme", and "Femme", with "Sans indication" selected. The "J'ai lu les conditions d'utilisation" field is a checkbox. Below the form is a toolbar with various icons.

# Éléments non sémantiques (`div` & `span`)

`<div>` (*block*) et `<span>` (*inline*)

- Pas de signification ni de rendu particulier par défaut
- Permettent de désigner ou grouper des fragments logiques du document
- Apparence et/ou comportement définis par CSS et/ou JavaScript

```
<div class="field">
  <label for="tos">J'accepte les conditions d'utilisation</label>
  <input id="tos" type="checkbox" />
</div>
```

⚠ Ne doivent pas remplacer des éléments plus adaptés.

- `<button onclick="...">Valider</button>` → ergonomie et accessibilité standards
- `<div class="button" onclick="...">Valider</div>` → tout à faire soi-même

# Accessibilité (a11y)

**Proposer un contenu lisible et utilisable aux personnes handicapées**, quel que soit ce handicap.

Obligation légale pour de nombreux services (pour la France, voir le [RGAA](#))



Un site accessible commence par un HTML bien structuré et bien décrit

## ARIA

Extension à HTML via des attributs **complémentaires** pour l'accessibilité ( `role` et `aria-*` )

```
<p role="alert" class="formMessage">Erreur de mot de passe</p>
```

```
<button aria-label="Fermer" onclick="...">X</button>
```

# DOM

---

## *Document Object Model*

- Résultat de l'interprétation du HTML
- Représentation en mémoire du document par le navigateur
- Expose une API permettant la lecture et la manipulation du document par des langages de script (en pratique, JavaScript)

L'inspecteur des outils de développement des navigateurs est une représentation du DOM.

```
<!DOCTYPE html>
<html class=" vzyuhmgea idc0_336" lang="en"> [event] [défilable]
  > <head> [ ] </head>
  > <body class="page-home">
    > <header id="mainHeader"> [flex]
      > <a id="homeLink" href="/" title="Home">
        
      </a>
      > <div id="mainHeaderText"> [flex]
        <div id="siteTitle">Benjamin Becquet</div>
      > <nav id="mainNav">
        > <ul id="insideLinks">
          > <li id="homeLinkMenu"> [ ] </li>
          > <li> [ ] </li>
            > espaces
          > <li> [ ] </li>
            > espaces
          > <li> [ ] </li>
        </ul>
        > <ul id="outsideLinks"> [ ] </ul>
      </nav>
    </div>
  </header>
  > <main>
    > <header> [ ] </header> [flex]
    > <p> [ ] </p>
    > <p>
      I do some
      <a href="/projects">open-source development</a>
      , most of it involving web and cartography.
    </p>
    > <p> [ ] </p>
      <h2>Interests</h2>
    > <p> [ ] </p>
```

# CSS

---

## *Cascading Style Sheets*

# Origine

Premiers temps du web : mélange contenu structuré + présentation dans le HTML

```
<body bgcolor="yellow">
  <h1 font="Comic Sans MS">titre</h1>
  <center><p>Lorem ipsum</p><center>
</body>
```

Problèmes vs. évolution rapide du web :

- Maintenabilité et passage à l'échelle
- Limitation d'expressivité
- Adaptabilité aux divers supports
- Accessibilité

→ Besoin d'un langage dédié à la présentation

# 1996 : Cascading Style Sheets

HTML

Contenu structuré

```
<html>
  <head></head>
  <body>
    <h1>titre</h1>
    <p>Lorem ipsum</p>
  </body>
</html>
```



CSS

Règles de présentation

```
body {
  background-color: yellow;
}
h1 {
  font-family: sans-serif;
}
p {
  text-align: center;
}
```

Version actuelle du standard : CSS 3.1

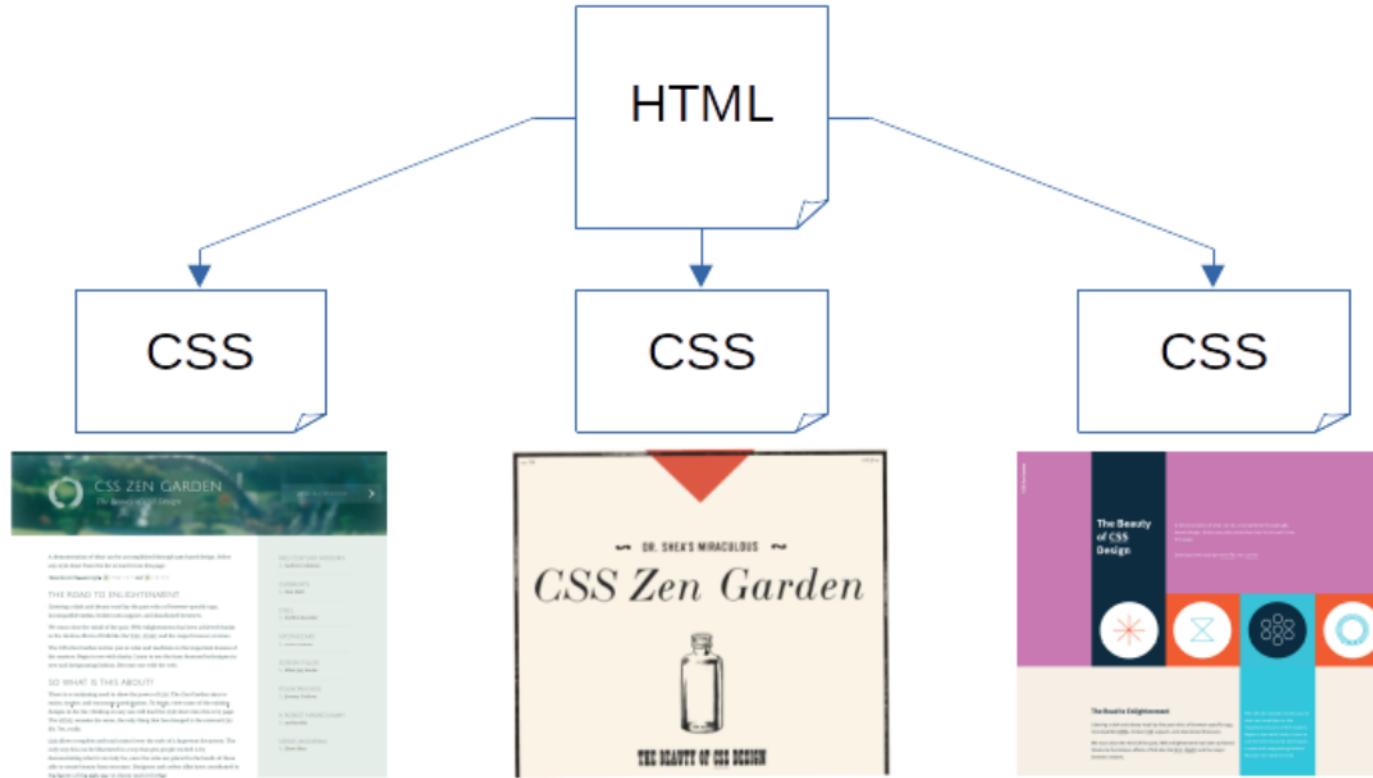
Référence sur le W3C : <https://www.w3.org/TR/CSS/>

Principes fondateurs similaires au HTML, notamment la **robustesse** et la tolérance aux erreurs

# Illustration

Bonne démonstration, ancienne mais toujours d'actualité : [CSS Zen Garden](#)

Une même page HTML pour de multiples feuilles de style au design très différent



# Lien HTML / CSS

- en référençant un fichier avec une balise `<link>` dans le `<head>`

```
<link rel="stylesheet" href="../../my-style.css" />
```

- dans le document, avec une balise `<style>`

```
<style>
  .banner {
    background-color: yellow;
  }
</style>
```

- inline*, spécifiquement pour un élément, avec l'attribut `style`

```
<button style="border:1px solid red;font-weight:bold">Annuler</button>
```

Les trois peuvent cohabiter.

# Syntaxe de base

```
sélecteur  
(, sélecteur*) {  
    propriété: valeur;  
    propriété: valeur;  
    ...  
}
```

**Sélecteur** : expression de ciblage d'éléments (ou parties) dans le document

**Propriété** : paramètre de style à appliquer aux éléments sélectionnés

**Valeur** : valeur possible définie par la règle

**Règle** : propriété + valeur

Exemple :

```
a {  
    color: #00bb76;  
    text-decoration: underline;  
}  
  
.bookName,  
.field label {  
    font-style: italic;  
}  
  
article p:first-of-type::first-letter {  
    font-size: 130%;  
}
```

# Sélecteurs principaux

Type	Syntaxe	Exemple
Universel	*	*
Type d'élément	<tag>	a
Identifiant	#<id>	#searchInput
Classe	.<class>	.message
Descendant	<ancestor> <child>	.message img
Enfant direct	<parent> > <child>	.message > img
Même parent	<child1> ~ <child2>	.message ~ .separator
Suivant	<child1> + <child2>	.message + .separator
Attribut	[<attr><operator><value>]	.message[lang=fr]

Référence MDN

# Sélecteurs: pseudo-classes ( : )

Ciblage d'un élément **en fonction de son "état"**.

- État dans le DOM :

:first-child , :first-of-type , :nth-child , ...

```
tr:nth-child(2n) {  
    background-color: pink;  
}
```

- États d'un lien :

:hover , :active , :visited , ...

```
a:hover {  
    text-decoration: none;  
}
```

- État de champs de formulaire :

:focus , :disabled , :invalid , :checked , ...

```
button:disabled {  
    opacity: 0.5;  
}
```

- Également opérateur :not

```
input:not(:checked) {  
    color: grey;  
}
```

Référence MDN

# Sélecteurs: pseudo-éléments ( :: )

Ciblage d'une **sous-partie** d'un élément.

- `::first-letter` , `::first-line`

```
p::first-letter {  
    font-size: 1.5em;  
    margin-left: 1em;  
}
```

Le chemin utilisateur se résument au texte brut, qui affiche sur une seule ligne du système d'ondlets pour visiter

- `::before` & `::after` : injection de contenu.

```
p:last-of-type::after {  
    content: "The end.>";  
    font-style: italic;  
    display: block;  
    text-align: right;  
}
```

isn't just a one-shot process, but a continuous  
the management of street name labels in a  
like [Mapbox GL](#). Maybe it's out the scope of this

*The end.*

Référence MDN

# Propriétés

---

Le standard définit :

- Quelles propriétés s'appliquent à quel type d'élément
- Quelles sont les types et valeurs possibles pour les propriétés
- Quelles propriétés sont héritées (ex : `font`) ou non (ex : `border`)

De nombreux usages :

- Typographie (`font-size`, `font-style`, `font-family`, ...)
- Couleurs, images, rendu (`color`, `background-color`, `background-image`, `filter`, ...)
- Marges internes/externes, bordures (`padding`, `margin`, `border`, ...)
- Dimensions (`width / height`, `min-width / max-height`, ...)
- Positionnement (`display`, `float`, `position`, ...)
- Transformations, animations, (`transition`, `transform`, ...)
- ...

[Référence MDN](#)

# Propriétés raccourcies

Certaines propriétés permettent de déclarer plusieurs valeurs à la fois.

C'est le cas de `font`, `background`, `margin`, `padding`, `border`, `flex`, ...

```
.example {  
  background-color: yellow;  
  background-image: url("imgs/bird.png");  
  background-repeat: no-repeat;  
  background-position: top left;  
}  
/* peut s'écrire */  
.example {  
  background: yellow url("imgs/bird.png") top left no-repeat;  
}
```

```
.example {  
  margin-top: 10px;  
  margin-right: 5px;  
  margin-bottom: 10px;  
  margin-left: 5px;  
}
```

=

```
.example {  
  margin: 10px 5px 10px 5px;  
}
```

=

```
.example {  
  margin: 10px 5px;  
}
```

# Unités de longueur

---

Les valeurs de tailles peuvent s'exprimer de très nombreuses façons :

## Absolues

- `px` : pixels
- `cm` , `mm` , `pt` , etc. : autres unités, non recommandées sur écran

## Relatives

- `%` : pourcentage (requiert une valeur de référence dans les parents)
- `em` : largeur d'un 'M' typographique, relatif au `font-size` parent
- `rem` (*root em*) : comme `em`, relatif au `font-size` racine
- `vw` / `vh` / `vmin` / `vmax` : pourcentages de hauteur/largeur d'affichage (*viewport*)

[Référence MDN](#)

# Flux de rendu et modèle de boîte

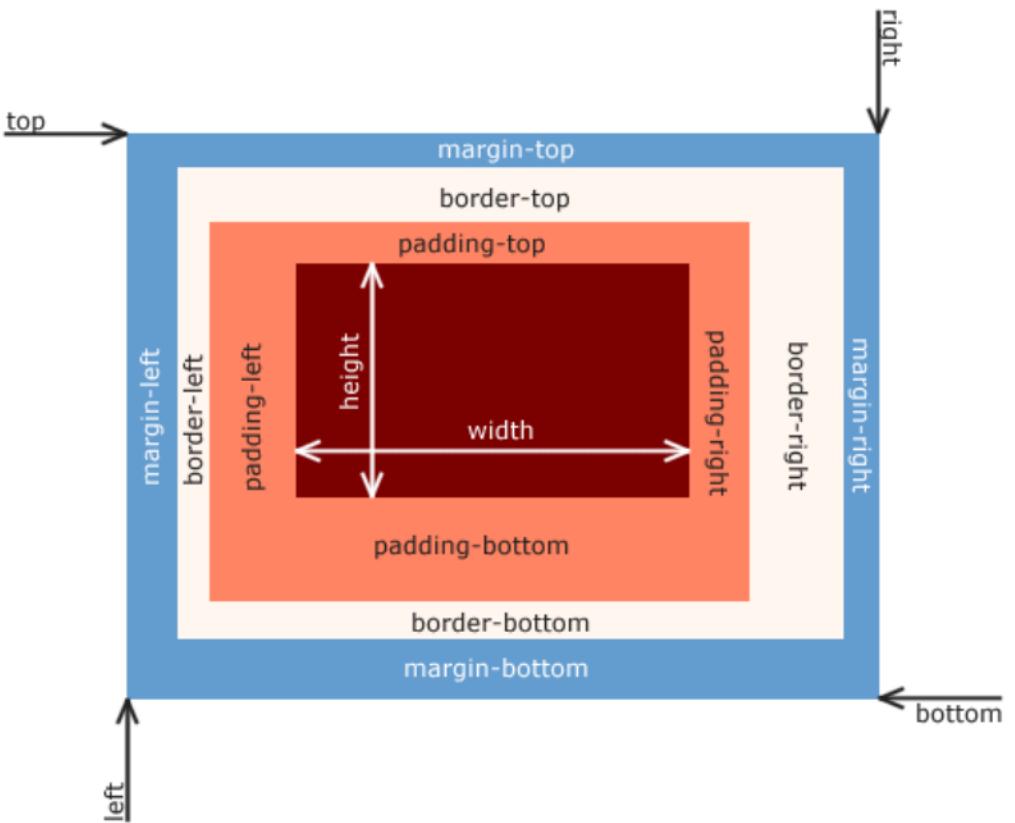
Deux grands types d'éléments en HTML/CSS :

- *block*:
  - 100% de la largeur
  - hauteur définie par son contenu
  - ex : `<p>` , `<article>` , `<div>` , ...
- *inline*:
  - largeur définie par son contenu
  - son rendu dans le flux du document
  - ex : `<a>` , `<em>` , `<input>` , ...

En CSS : propriété `display`

(qui peut valoir `inline-block` et plein d'autres valeurs...)

Le *box model* s'applique aux éléments block



(Source image [WikiPedia](#))

# Layout

---

CSS offre maintenant des solutions matures pour gérer des arrangements complexes :

## Flexbox

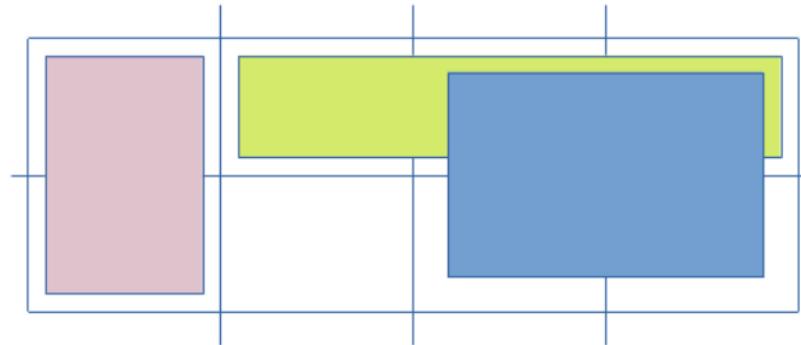


`display: flex;`

- juxtapositions
- centrages et alignements
- designs fluides

[A Complete Guide to Flexbox](#)

## Grid



`display: grid;`

- templates assimilables à une grille
- facilement réorganisable
- overlap possible

[A Complete Guide to Grid](#)

► Flexbox vs. CSS <https://www.youtube.com/watch?v=hs3piaN4b5I>

## **Media queries: @media**

Application de sélecteurs selon les caractéristiques et fonctionnalités du client :

```
@media print {  
    video {  
        display: none;  
    }  
}  
  
@media (prefers-color-scheme: dark) {  
    body {  
        background-color: black;  
        color: white;  
    }  
}  
  
@media (prefers-reduced-motion) { ... }  
  
@media (pointer: none) { ... }
```

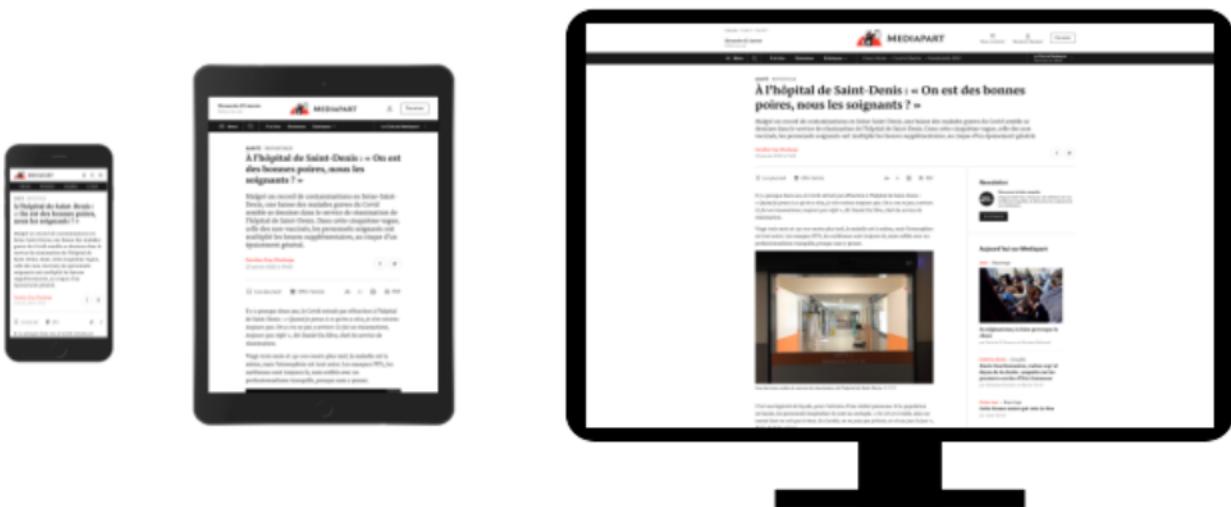
Référence MDN

# Responsive Web Design

Application commune des *media queries*.

Principe :

- servir un même contenu HTML quelque soit le *device*
- en fonction des caractéristiques du navigateur, adapter le contenu (déplacer/redimensionner/masquer/...) via CSS
- adaptable à largeur/hauteur d'écran, mais aussi orientation (portrait/paysage), finesse d'affichage, etc...



```
body {  
    font-size: 14px;  
}  
  
/* "breakpoint" */  
@media (min-width: 600px) {  
    body {  
        font-size: 16px;  
    }  
}  
  
@media (min-width: 1000px) {  
    body {  
        margin: 0 auto;  
        max-width: 800px;  
    }  
}
```

Approche *mobile-first*:

- Mobile = cas par défaut
- On surcharge ensuite pour les grands écrans

## ***Custom properties (aka. "variables CSS")***

- Apparues avec CSS3
- Permettent de définir des valeurs réutilisables
- Sont héritées

```
:root {  
  --mainColor: black; /* Déclaration */  
}  
  
aside {  
  --mainColor: #333;  
}  
  
.note {  
  color: var(--mainColor); /* Accès */  
  background-color: var(--bgColor, white); /* Accès avec valeur par défaut */  
}
```

[Référence MDN](#)

# "Cascading" ?

La "cascade" = **algorithme au cœur du CSS** qui calcule **quelles règles de style s'appliquent réellement aux éléments du DOM**, à partir de différentes règles contradictoires.

Critères :

## 1. Importance de la règle

Règle de transition > règle marquée `!important` > règle normale

## 2. Origine de la règle

Site web > feuille de style utilisateur > style par défaut du navigateur

## 3. Spécificité des sélecteurs (Réf. MDN)

*inline* dans l'attribut HTML `style` > `#id` > `.class` > type

Précision (ex : `main p > p`)

## 4. Ordre de déclaration

À score égal par ailleurs, la dernière règle définie s'applique



▲ Bonne synthèse  
<https://wattenberger.com/blog/css-cascade>

# Problématiques du CSS sur les projets d'envergure

---

- Standard **très** vivant, en évolution constante

Voir [les évolutions en 2022](#)

En pratique, dépend du support par les navigateurs (voir <https://caniuse.com/>)

Comment continuer à supporter les vieilles versions ?

- Problématiques réseau

Comment ne pas envoyer du style inutile ?

Comment s'assurer qu'il est à jour par rapport au HTML ?

- Complexité de la cascade, surcharges de règles

Pratiques de *namespacing* des sélecteurs ([OOCSS, BEM, etc...](#))

```
.search-box__button_big { ... }
```

→ Pour les gros projets, il est rare d'écrire du CSS brut et *from scratch*.

# Pratiques et outils actuels

---

- Préprocesseurs ou transpileurs (ex : [Sass](#), [PostCSS](#))
  - Surcouche au langage, transformé en vrai CSS
  - Support syntaxique plus large (ex : boucles, *nesting* de sélecteurs)
  - Application de fonctions et automatisation à la compilation (ex : préfixage de propriétés)
- Frameworks et *design systems* clef-en-main
  - Ex : [Bootstrap](#), ensemble de classes utilitaires de haut niveau

```
<button class="btn btn-lg btn-primary">
```
  - Ex : [Tailwind](#), [Tachyons](#), ensemble de classes utilitaires "atomiques" + compilation

```
<button class="m-10 bg-red-300 text-white px-4 py-2 text-xs shadow-lg">
```
- Lien avec le code JavaScript, approches "CSS-in-JS"
  - *namespacing* automatique via le code JavaScript

```
<button class="sc-bkzYnD sc-dmlqKv lg00oH gNtXwi"> 🤔
```

## Remarques générales

---

## À retenir

---

Ce cours est une introduction très rapide !

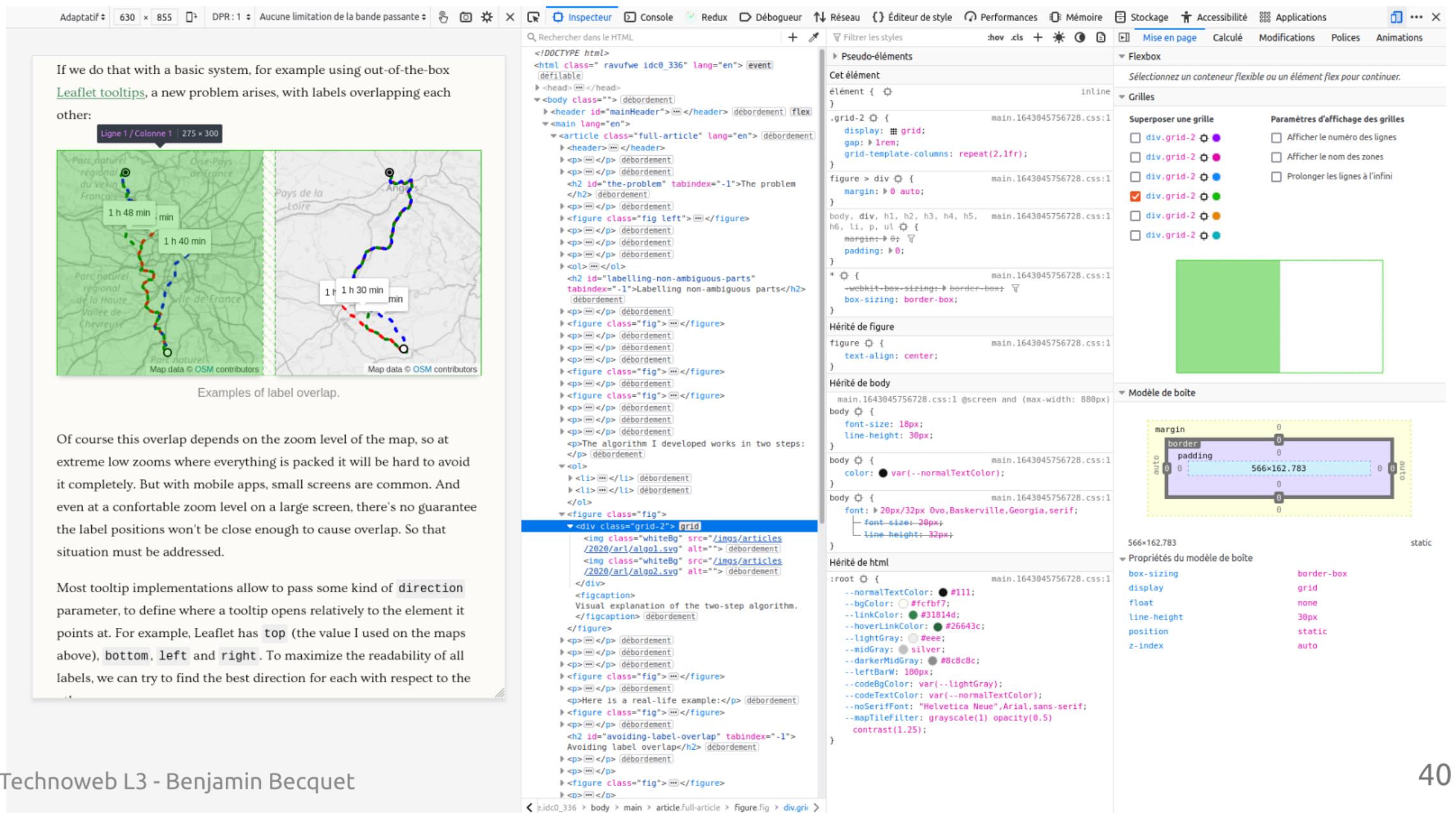
HTML et CSS sont trop complexes pour tout connaître d'un coup.

Plus important :

- Connaître les grands principes des langages
- Être sensible aux bonnes pratiques (standards, accessibilité, etc.)
- Savoir chercher l'information
- Se tenir un minimum au courant des évolutions
- Technologie toujours "bidouillable" → bricoler !

# Outils de développement des navigateurs

If we do that with a basic system, for example using out-of-the-box [Leaflet tooltips](#), a new problem arises, with labels overlapping each other:



Ligne 1 / Colonne 1 | 275 x 300

Parc naturel régional du Vexin Français  
Oise-Pays de France  
Pays de la Loire  
Angers  
Paris  
Map data © OSM contributors

Examples of label overlap.

Of course this overlap depends on the zoom level of the map, so at extreme low zooms where everything is packed it will be hard to avoid it completely. But with mobile apps, small screens are common. And even at a confortable zoom level on a large screen, there's no guarantee the label positions won't be close enough to cause overlap. So that situation must be addressed.

Most tooltip implementations allow to pass some kind of `direction` parameter, to define where a tooltip opens relatively to the element it points at. For example, Leaflet has `top` (the value I used on the maps above), `bottom`, `left` and `right`. To maximize the readability of all labels, we can try to find the best direction for each with respect to the

40

# Pour finir

---

Deux références que j'aime bien :



*Resilient Web Design*

The screenshot displays a blog post from a website. At the top, there is a purple banner with the text "24 JOURS" and "WEB". The main title of the post is "QUELQUES TECHNIQUES HTML ET CSS POUR RÉDUIRE L'UTILISATION DE JAVASCRIPT". Below the title, the author is listed as "Anthony Ricaud" and the date as "22 décembre 2020". A small note indicates it's a 12-minute read. The post begins with a paragraph about the increasing use of JavaScript and its disadvantages. It then lists three bullet points: "• temps de chargement allongés ;", "• sites inutilisables si le code JavaScript provoque des erreurs ou n'arrive pas à se télécharger ;", and "• ergonomie, réactivité et accessibilité laissant à désirer sans une équipe ayant les moyens d'y faire attention.". A summary paragraph follows, mentioning native solutions provided by browsers. At the bottom, a note states that the post will explore these native solutions.

**QUELQUES TECHNIQUES HTML ET CSS POUR RÉDUIRE L'UTILISATION DE JAVASCRIPT**

Anthony Ricaud  
22 décembre 2020  
12 minutes

**D**e plus en plus de sites web se reposent sur JavaScript pour la majorité des interactions mises à disposition. Cela permet de créer des expériences fort attrayantes mais cela vient aussi quelquefois avec des effets indésirables :

- temps de chargement allongés ;
- sites inutilisables si le code JavaScript provoque des erreurs ou n'arrive pas à se télécharger ;
- ergonomie, réactivité et accessibilité laissant à désirer sans une équipe ayant les moyens d'y faire attention.

Au vu de ces inconvénients, se reposer sur les solutions proposées nativement par les navigateurs permet de bénéficier à peu de frais de l'expertise de la communauté créant les standards du web. Ces solutions ont généralement l'avantage d'utiliser moins de code, réduisant ainsi les efforts de maintenance pour une équipe de développement (par exemple, pas besoin de mettre à jour les librairies utilisées).

Dans cet article, nous allons explorer certaines de ces solutions natives

[Quelques techniques HTML et CSS pour réduire l'utilisation de JavaScript](#)