

# Initiation au développement web

Créer des pages web avec HTML & CSS

# Historique

## Le World Wide Web

Inventé vers **1990** par **Tim Berners-Lee** dans le cadre de son travail au **CERN**

Souvent confondu avec **Internet**, il en est en réalité l'une des nombreuses applications, parmi lesquelles :

- Le courrier électronique,
- La visio-conférence,
- Le partage de fichiers en pair-à-pair (ou peer-to-peer).

# Historique

## Le World Wide Web

A la convergence de trois technologies :

- **URL** (Uniform Resource Locator), qui gère l'adressage sous une forme standardisée vers une ressource du web.
- **HTTP** (HyperText Transfert Protocol), qui permet la **communication serveur-client**.  
Les clients HTTP les plus connus du grand public sont les **navigateurs web**, mais il en existe d'autres (robots d'indexation, agrégateurs de flux de syndication...)
- **HTML** (HyperText Markup Language), qui permet l'écriture de **documents hypertextuels**, reliant différentes ressources du Web entre elles.

# Historique

## HTML

- **HyperText** fait référence à la capacité d'HTML à faire des **liens entre les documents**,
- **Markup** correspond à la capacité d'annoter du texte brut grâce à des **balises**, afin d'indiquer au navigateur comment traiter l'affichage de ce texte,
- **Language** car il s'agit d'un **langage informatique**.

# Historique

## HTML

Principes fondateurs :

- **Ouvert** : ses spécifications techniques sont publiques, sans restriction d'accès ou de mise en œuvre,
- **Sémantique** : il se concentre sur le sens des contenus plutôt que sur leur présentation,
- **Résilient** : un document mal formaté ne "plante" pas, même s'il peut y avoir des effets de bord indésirables.

# Historique

## HTML

- Au début : pas de normes, son usage est dicté par l'implémentation des navigateurs.
- 1994 : Tim Berners-Lee quitte le CERN et fonde le **W3C**, dont le but est de promouvoir la compatibilité des technologies du Web.
- 1997 : le W3C publie ses recommandations pour **HTML 3.2** puis **HTML 4**, puis base ses spécifications sur le format XML nouvellement standardisé, afin de développer **le format XHTML**.

# Historique

## HTML

- 2004 : création du **WHATWG** (Web Hypertext Application Technology Working Group) par certains fabricants de navigateurs web (Mozilla pour Firefox, Opera, et Apple pour Safari), avec une approche plus pragmatique et centrée sur les implémentations existantes.
- 2007 : les travaux du WHATWG sont adoptés par le W3C comme point de départ à la spécification **HTML 5**
- 2011 : des différences de points conduisent le WHATWG à créer **HTML Living Standard**, une spécification prévue pour évoluer constamment, en fonction des évolutions fonctionnelles rapides des navigateurs.
- 2019: W3C et le WHATWG signent un mémorandum actant notamment leur travail conjoint sur ce standard évolutif.

# Historique

## CSS

CSS est l'abréviation de **Cascading StyleSheets**, c'est à dire Feuilles de Styles en Cascade

À l'origine, le HTML combinait la **structuration du contenu** et la **description de la présentation** :

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



# Historique

## CSS

L'évolution constante du web pousse vers l'adoption d'un **langage dédié à la présentation**, en fonction de plusieurs enjeux :

- **Maintenabilité et passage à l'échelle** : l'augmentation de la taille et de la complexité des sites et applications web rend la maintenance du code HTML (corrections, mises à jours et évolutions de fonctionnalités) de plus en plus difficile ;
- **Adaptabilité aux divers supports** : l'existence de plusieurs types de médias de restitution (impression, écran, télévision...) demande la possibilité d'une adaptation plus fine aux contraintes de chacun ;
- **Accessibilité** : en fonction de leurs besoins et contraintes, les utilisateurs configurent des préférences sur leurs terminaux, qu'il est intéressant de pouvoir répercuter sur les sites web.

# Historique

## CSS

Tout comme HTML, le CSS est **un langage ouvert et résilient**.

Il est développé par "**niveaux**", ce qui contraint chaque nouvelle itération à se baser sur la précédente, et à être **rétro-compatible** avec elle.

Créer des pages HTML  
sémantiques

# Structure d'un document HTML

Éléments, balises et attributs

Un **document HTML** est une **arborescence** d'éléments, aussi appelé **DOM** (Document Object Model), et composé de **nœuds** :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <title>Page d'exemple</title>
  </head>
  <body>
    <h1>Page d'exemple</h1>
    <p>Ceci est <a href="exemple.html">un exemple de lien</a>.</p>
  </body>
</html>
```

# Structure d'un document HTML

## Éléments, balises et attributs

Un **élément** est généralement formé d'une **balise ouvrante** et d'une **balise fermante**, encadrant du **contenu**.

Les éléments sont structurés de manière **hiérarchique**, chacun doit être intégralement contenu par son **ancêtre** (ou parent), sans chevauchement. Il est impératif de refermer les balises dans l'**ordre inverse** de leur ouverture.

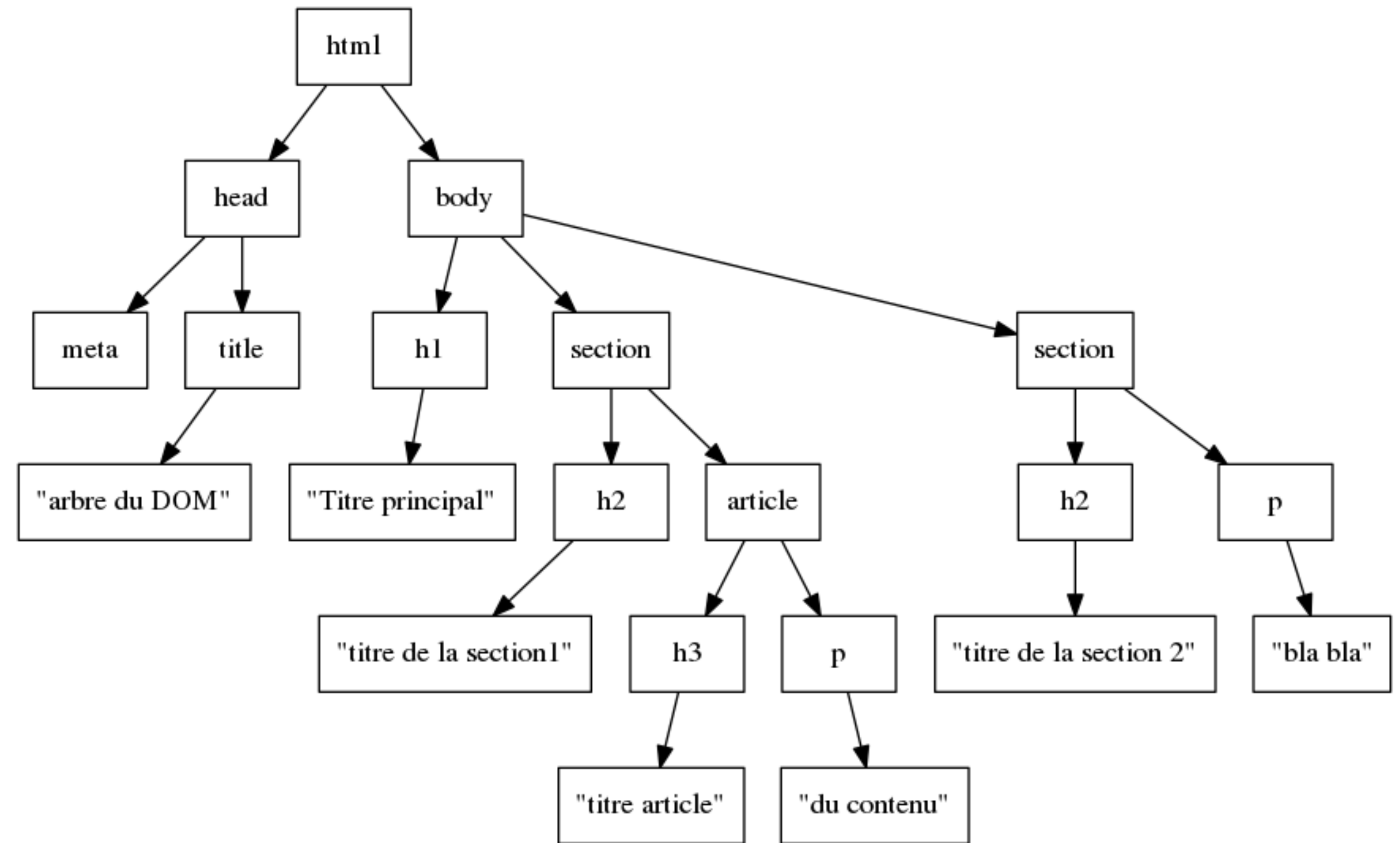
Certains éléments peuvent être **auto-fermants** : `<br />`

# Structure d'un document HTML

Éléments, balises et attributs

Le **nœud-racine** d'un document HTML est toujours la balise `<html>`, qui possède elle-même deux nœuds **descendants** (ou enfants) : `<head>` et `<body>`.

Les nœuds sans descendants sont appelés des **feuilles**.



# Structure d'un document HTML

Éléments, balises et attributs

Un **attribut** sert à apporter des informations supplémentaires sur un élément :

```
<a href="exemple.html" > ... </a>
```

Certains attributs peuvent être **vides** :

```
<input name="nom" disabled />
```

# Structure d'un document HTML

Éléments, balises et attributs

Enfin, il est possible d'ajouter des **commentaires**, qui ne seront pas affichés dans le rendu final, mais permettent d'organiser le code et d'indiquer des précisions pour en faciliter la maintenance :

```
<!-- Le contenu de mon commentaire -->
```



# Structure d'un document HTML

## Les éléments obligatoires

- Les éléments `<!DOCTYPE html>` et `<html>` indiquent au navigateur que le document est rédigé en HTML.  
L'attribut `lang` sur la balise `<html>` permet en outre de préciser la **langue du document**.
- L'élément `<head>` contient le **titre** de la page, les **ressources annexes** (ex. icône, appels de feuilles de styles CSS ou de scripts Javascript externes) et ses **métadonnées** (utiles notamment pour le SEO et le partage sur les réseaux sociaux)...
- L'élément `<body>` contient le **contenu** de notre document.

# Structure d'un document HTML

## Les éléments de section

Ces éléments ont une valeur sémantique, et permettent de mieux structurer la page :

- L'élément `<header>` (à ne pas confondre avec `<head>` !) signale l'**entête** de la page,
- L'élément `<nav>` regroupe les **principaux liens de navigation**,
- Les éléments `<section>` indiquent un regroupement suivant **un sujet ou une fonctionnalité commune**,
- L'élément `<article>` correspond à du **contenu indépendant**, pouvant être retiré du contexte de la page (ex. via un flux RSS) sans nuire à sa compréhension,
- L'élément `<aside>` signale un **contenu annexe ou complémentaire**,
- L'élément `<footer>` indique le **pied de page**.

# Contenus de base

## Titres et paragraphes

Les éléments `<h1>` à `<h6>` représentent les **niveaux de titres** d'un document, du plus important au moins important.

Un document ne peut comporter qu'**un seul titre de niveau 1**.

Le **contenu textuel** est contenu par un élément `<p>`.

La balise auto-fermante `<br />` permet le **saut de ligne**.

# Contenus de base

## Liens

Il s'agit du principe de base du web : faire des liens vers d'autres pages.

On utilise la balise `<a>` avec l'attribut `href` pour préciser l'**URL du lien**, c'est-à-dire sa **cible**.

```
<a href="autre-page.html">Texte du lien</a>
```

L'attribut `target="blank"` permet d'ouvrir le lien dans un nouvel onglet.

```
<a href="http://www.google.fr" target="_blank">Un lien vers Google</a>
```

# Contenus de base

## Liens

On peut également utiliser des liens internes au document, appelés **ancres**, permettant de cibler un élément précis, identifié grâce à son attribut `id`.

```
<a href="#monTitre">Lien vers mon titre de niveau 2</a>
```

```
...
```

```
<h2 id="monTitre">Mon titre de niveau 2</h2>
```

# Contenus de base

## Images

On utilise la balise auto-fermante `<img />`

```

```

L'attribut `src` permet de préciser l'**URL** (relative ou absolue) de l'image.

L'attribut `alt` est obligatoire, il permet de **décrire le contenu** de l'image.

Il est affiché si le lien de l'image ne fonctionne pas, ou par certaines technologies d'assistance.

# Contenus de base

## Listes

Il existe deux types de listes :

- Les **listes ordonnées**, qui sont numérotées et déclarées par la balise `<ol>`
- Les **listes non-ordonnées**, non-numérotées et déclarées par la balise `<ul>`

Chaque élément de la liste est contenu dans une balise `<li>`

```
<h1>Mon tutoriel</h1>
<ol>
  <li>Ma première étape</li>
  <li>Ma deuxième étape</li>
</ol>
```

# Contenus de base

## Listes

Les listes peuvent être **imbriquées**.

```
<ul>
  <li>Un élément :
    <ul>
      <li>Un sous-élément</li>
      <li> Un autre sous-élément</li>
    </ul>
  </li>
  <li>Un autre élément</li>
</ul>
```



# Contenus avancés

## Tableaux

Un tableau est un **ensemble structuré de données** présentées en lignes et en colonnes.  
Il permet de retrouver facilement des valeurs au croisement de plusieurs types de données.

<tr> (ligne)	Nom	Famille		Statut de conservation
	Macareux	Alcidae		Vulnérable
	Mésange bleue	Paridae		Préoccupation mineure
	Mouette tâchetée	Laridae		Préoccupation mineure

# Contenus avancés

## Tableaux

Il est constitué de :

- Une balise `<table>` englobant l'**ensemble du tableau**, et contenant :
  - Une balise `<caption>` contenant la **légende**,
  - Une ou plusieurs balises `<tr>` délimitant chacune des **lignes**, et contenant :
    - Une ou plusieurs balises `<th>` précisant les **entêtes** des colonnes et des lignes,
    - Une ou plusieurs balises `<td>` contenant les **données** elles-même,
  - Optionnellement, les balises `<tr>` peuvent être incluses dans des éléments `<thead>`, `<tbody>` et `<tfoot>` pour délimiter les différentes **parties du tableau** - respectivement **entête, corps et pied**. Ces éléments n'ont pas directement d'effet en HTML, mais peuvent servir de points d'accroche pour les styles CSS.

# Contenus avancés

## Tableaux

Dépenses en 2024	
Éléments	Coût
Papeterie	2 000 €
Pâtisserie	6 000 €
Total	8 000 €

```
<table>
  <caption>Dépenses en 2024</caption>
  <thead>
    <tr>
      <th>&Eacute;léments</th>
      <th>Coût</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>Papeterie</th>
      <td>2 000 €</td>
    </tr>
    <tr>
      <th>Pâtisserie</th>
      <td>6 000 €</td>
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <th>Total</th>
      <td>8 000 €</td>
    </tr>
  </tfoot>
</table>
```

# Contenus avancés

## Formulaires

Les formulaires permettent à l'utilisateur d'**interagir** avec le site web via l'**envoi de données**. Ces données sont ensuite traitées côté serveur, par exemple grâce à PHP ou Java.

Ils sont constitués :

- d'une balise `<form>` englobant l'**ensemble des éléments** du formulaire,
- d'un ou plusieurs **éléments de formulaire** `<input>`, `<select>` ou `<textarea>`,
- d'une balise `<button>` permettant l'**envoi du formulaire**.

# Contenus avancés

## Formulaires > Les éléments de formulaire

L'élément le plus simple est le **champ de texte** :

```
<input type="text" />
```

Cependant, l'attribut `type` peut prendre d'autres valeurs :

- Certaines permettent d'ajouter des **contraintes de validation** gérées directement par le navigateur. Par exemple : `email`, `tel`, `url`...
- D'autres modifient l'interface de manière plus importante :
  - Le **champ de mot de passe** : `<input type="password" />`
  - Le **champ de sélection de fichier** : `<input type="file" accept="image/png, image/jpeg" />`



# Contenus avancés

## Formulaires > Les éléments de formulaire

L'élément `<textarea>` permet d'éditer du **texte sur plusieurs lignes**.

On peut préciser le **nombre de lignes affichées** grâce à l'attribut `rows` : `<textarea rows="5">`

L'élément `<select>` permet de créer une **liste déroulante**.

Il englobe des éléments `<option>` correspondant chacun à **un choix de la liste**.

Chacun de ses choix :

- comporte un attribut `value` précisant la **valeur** qui sera envoyé au site via le formulaire,
- contient le **texte** qui sera restitué à l'utilisateur par l'interface.

```
<select name="animaux">
  <option value="chien">Chien</option>
  <option value="chat">Chat</option>
  <option value="hamster">Hamster</option>
</select>
```

# Contenus avancés

## Formulaires > Les éléments de formulaire

La **case à cocher** : `<input type="checkbox" />`

- Elle peut être unique,
- On peut en sélectionner plusieurs à la fois,
- On peut la décocher.

☐ J'accepte les [CGV](#)

Le **bouton radio** : `<input type="radio" />`

- Il y en a forcément deux ou plus,
- On peut en sélectionner un seul à la fois.

☐  Mastercard

☐  Visa

# Contenus avancés

## Formulaires > Les éléments de formulaire

Les cases à cocher et les boutons radio doivent forcément comporter :

- Un attribut **id** identique à l'attribut `for` de leur libellé (`<label>`) associé,
- Un attribut **value** indiquant leur valeur, qui sera envoyé avec le formulaire,
- Un attribut **name** afin de les relier, mais attention :
  - les attributs `name` de tous les boutons radio / cases à cocher appartenant à la même catégorie doivent être identiques (= regroupement programmatique),
  - Il n'est pas possible de mixer des boutons radio et des cases à cocher dans une même catégorie.

Enfin, ils peuvent optionnellement recevoir un attribut **checked** afin d'être cochés par défaut.



# Contenus avancés

## Formulaires > Les éléments de formulaire

Chacun de ces éléments doit être associé à un **libellé**, contenu dans une balise `<label>`

Cet intitulé est relié au champ qui lui correspond via un attribut `for`, qui reprend la valeur de l'attribut `id` de ce champ.

```
<label for="prenom">Prénom</label>  
<input type="text" id="prenom" />
```

# Contenus avancés

## Formulaires > Les regroupements de champs


Il est parfois nécessaire d'indiquer le **regroupement sémantique** de plusieurs champs de formulaire.


On réalise cela grâce à la balise `<fieldset>`, qui doit commencer par un élément `<legend>` pour indiquer la **nature** (ou le titre) du regroupement.

Les `<fieldset>` peuvent être **imbriqués**.

Informations de paiement

Type de paiement

☐  Mastercard

☐  Visa

Carte bancaire

Numéro de carte

Nom du porteur

Code de sécurité

# Contenus avancés

## Formulaires > Les regroupements de champs

Un cas classique est celui des boutons radio.

```
<fieldset>
  <legend>Choisissez votre animal préféré</legend>

  <input type="radio" name="animal" value="chien" id="chien" />
  <label for="chien">Chien</label>

  <input type="radio" name="animal" value="chat" id="chat" />
  <label for="chat">Chat</label>

  <input type="radio" name="animal" value="hamster" id="hamster" />
  <label for="hamster">Hamster</label>
</fieldset>
```

Dans ce cas, le **regroupement** est également signalé au navigateur **de manière programmatique** via l'attribut `name`, dont la valeur est identique pour tous les boutons radio faisant partie du même regroupement.

# Contenus avancés

## Formulaires > Les regroupements de champs

Autre exemple de regroupement, uniquement sémantique :

```
<fieldset>
  <legend>Civilité</legend>

  <label for="nom">Nom</label>
  <input type="text" id="nom" />

  <label for="prenom">Prénom</label>
  <input type="text" id="prenom" />
</fieldset>
```

# Contenus avancés

## Formulaires > Les boutons

L'élément `<button>` a pour type implicite `submit`, qui permet la **soumission du formulaire** au clic.

```
<button>Envoyer votre message</button>
```

Il est cependant possible de lui indiquer un type `button`, qui n'a pas de comportement par défaut mais permet d'attacher des comportements scriptés en Javascript.

```
<button type="button">Ajouter aux favoris</button>
```

# Les balises génériques

Elles permettent de **regrouper des éléments** afin de leur appliquer du **CSS commun**, sans ajouter de valeur sémantique.

Elles sont de deux types :

- Les balises de type **block** : `<div>`, qui prennent toute la largeur disponible dans la page et provoquent un retour à la ligne,
- Les balises de type **inline** : `<span>`, qui ne prennent que la largeur nécessaire à leur contenu, et ne provoquent pas de retour à la ligne.

Attention : on peut mettre une balise de type inline à *l'intérieur* d'une balise de type block, mais pas l'inverse.

Styler en CSS



# Notions de base

## La séparation des préoccupations

La séparation du fond - en HTML - et de la présentation - en CSS - facilite l'**adaptation de l'affichage des documents web** en fonction de plusieurs paramètres, notamment :

- Les **préférences utilisateur** (ex. thème foncé, contrastes, tailles de polices...),
- Le **type de terminal client** (ex. ordinateur, mobile...),
- La **résolution d'écran**,
- La compatibilité entre les navigateurs...



# Notions de base

## La séparation des préoccupations

Il est possible d'ajouter du CSS à un document HTML de plusieurs façons :

- via un **attribut** `style` sur les balises HTML dont on souhaite modifier la présentation,
- contenues dans un **élément** `<style>...</style>`, en général lui-même contenu dans l'élément `<head>` du document,
- dans un **fichier externe** portant l'**extension de fichier .css**, importé dans le document HTML via l'élément `<link>` :

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

# Notions de base

## La séparation des préoccupations

La première méthode est verbeuse peu pratique à maintenir, et la seconde ne permet d'agir que sur un seul document à la fois - alors qu'un site web comporte en général plusieurs pages.

C'est pourquoi en dehors de certains cas d'usage, on préférera généralement la troisième méthode, qui permet d'appliquer les mêmes feuilles de styles à plusieurs documents, ce qui garantit une **cohérence de design** à l'échelle du site web.

# Notions de base

## Structure d'une règle CSS

Un **sélecteur** permet le ciblage d'un ou plusieurs éléments (ou parties d'élément) dans le document.

Une **propriété** est un paramètre de style appliqué aux éléments sélectionnés, qui peut prendre une valeur.

La combinaison d'une propriété et d'une valeur est appelée **déclaration**.

# Notions de base

## Structure d'une règle CSS

Une **règle** consiste en une ou plusieurs déclarations appliqués à un ou plusieurs sélecteurs suivant la syntaxe suivante :

```
selecteur  
(, selecteur) {  
    propriété: valeur;  
    (propriété: valeur;  
}
```

# Notions de base

## La cascade

Il s'agit de l'**algorithme** au coeur de CSS, qui calcule quelles déclarations de styles s'appliquent réellement aux éléments HTML.

Différents critères modifient le "**poids**" d'une déclaration :

- L'**importance** : l'ajout de `!important` à la fin d'une déclaration en augmente le poids ;
- L'**origine** :
  - le **site web** (les seuls sur lesquels vous pouvez intervenir),
  - la **feuille de styles de l'utilisateur**,
  - les **styles par défaut du navigateur** ;
- La **spécificité du sélecteur** (nous y reviendrons) ;
- La **position** de la règle : à score égal pour les critères précédents, c'est la **dernière déclaration définie** qui s'applique.

# Notions de base

## L'héritage

Certaines propriétés dont la valeur est déclarée pour un élément donné peuvent être **héritées** par les éléments **descendants** (ou enfants) de celui-ci, tandis que d'autres ne le sont pas.

Par exemple, les propriétés de couleurs ou de polices seront héritées - afin de réduire la quantité de CSS nécessaire à l'affichage - tandis que celles de largeur ou de hauteur ne le seront pas - car cela engendrerait au contraire une complexification inutile.

# Notions de base

## Flux de rendu et modèle de boîtes

Le modèle évoqué pour `div` et `span` concerne en réalité **tous les éléments HTML**.

Les élément de type **block** :

- occupent toute la largeur disponible et provoquent un retour à la ligne,
- les propriétés de largeur et de hauteur sont appliquées,
- les propriétés de marges et de bordures repoussent les éléments alentours,
- ex. `<p>`, `<article>`...



# Notions de base

## Flux de rendu et modèle de boîtes

Les éléments de type **inline** :

- n'occupent que la largeur nécessaire à l'affichage de leur contenu et ne provoquent pas de retour à la ligne,
- les propriétés de largeur et de hauteur ne sont pas appliquées,
- les propriétés de marges et de bordures sont appliquées mais ne repoussent pas les éléments alentours;
- ex. `<a>`, `<input>`...

# Notions de base

## Flux de rendu et modèle de boîtes

Il est cependant possible de changer le type d'un élément via la propriété `display`, qui peut donc prendre comme valeur `block` ou `inline`, mais également `inline-block`, afin de créer un **type hybride**, qui possède des caractéristiques des deux types :

- comme `inline`, il n'occupe que la largeur nécessaire à son contenu et ne provoque pas de retour à la ligne,
- comme `block`, les propriétés de largeur et de hauteur seront appliquées, et les propriétés de marges et de bordures auront un effet sur les éléments voisins.

# Les sélecteurs

## Attributs HTML

Il est possible de définir des "**points d'accroche**" sur un élément HTML grâce à deux types d'attributs :

- L'attribut `id` :
  - ne contient qu'**une seule valeur**,
  - cette valeur est **unique** à l'élément qui la porte,
  - il peut être **utilisé par le HTML** lui-même : liens internes (ancres), libellés de champs...
- L'attribut `class` :
  - un élément peut porter **plusieurs valeurs**,
  - une valeur peut être assignée à **plusieurs éléments**,
  - il n'est utilisé que pour la **sélection via CSS**.

# Les sélecteurs

## Attributs HTML

Chaque valeur peut être composée de lettres (majuscules et minuscules), de chiffres et/ou de tirets, mais doit impérativement commencer par une lettre.

# Les sélecteurs

## Types de sélecteur

### Les **sélecteurs simples** :

- Sélecteur **universel** : \*
- Par élément : <balise>
- Par **class** : .<valeur de l'attribut class>
- Par **id** : #<valeur de l'attribut id>

```
* {  
    font-style: italic;  
}
```

```
p {  
    color: red;  
}
```

```
.myClass {  
    text-decoration: underline;  
}
```

```
#myId {  
    font-family: monospace;  
}
```

# Les sélecteurs

## Types de sélecteurs

Ces différents sélecteurs peuvent être combinés entre eux, pour former des **sélecteurs composés** :

```
p.myClass#myId {  
  font-size: 1.5rem;  
}
```

# Les sélecteurs

## Types de sélecteurs

Les **sélecteurs d'attributs** permettent de cibler un élément en fonction de la présence d'un attribut sur sa balise, et éventuellement de la valeur de cet attribut.

```
a[href], input[type="checkbox"]
```

Il est possible d'effectuer un filtrage plus fin de la valeur via des opérateurs logiques :

[https://developer.mozilla.org/fr/docs/Web/CSS/Attribute\\_selectors](https://developer.mozilla.org/fr/docs/Web/CSS/Attribute_selectors)



# Les sélecteurs

## Types de sélecteurs

Les **combinateurs CSS** permettent de créer des **sélecteurs complexes** :

- Le combineur de **descendance**, qui cible les éléments correspondants au deuxième sélecteur uniquement s'ils ont un ancêtre qui correspond au premier sélecteur :

```
ul.ma-liste li
```

- Le combineur de **descendance directe**, qui cible les éléments correspondant au deuxième sélecteur uniquement s'ils sont les descendants *directs* d'un élément correspondant au premier sélecteur :

```
ul.ma-liste > li
```

# Les sélecteurs

## Types de sélecteurs

Les **combinateurs de voisinage** :

- Le combineur de **voisin suivant**, qui cible les éléments correspondant au 2e sélecteur uniquement s'ils se trouvent après un élément correspondant au 1er sélecteur :



The image shows a CSS selector example for the 'voisin suivant' (next) combinator. It consists of the text 'img ~ p' in a monospaced font, with 'img' and 'p' in red and the tilde '~' in white. The text is centered within a dark rectangular box.

- Le combineur de **voisin immédiat**, qui cible les éléments correspondant au deuxième sélecteur uniquement s'ils se trouvent *immédiatement* après un élément correspondant au premier sélecteur :



The image shows a CSS selector example for the 'voisin immédiat' (adjacent) combinator. It consists of the text 'img + p' in a monospaced font, with 'img' and 'p' in red and the plus sign '+' in white. The text is centered within a dark rectangular box.

# Les sélecteurs

## Types de sélecteurs

Les **pseudo-classes** permettent de cibler un élément en fonction de :

- sa **position dans le DOM** : `:first-child`, `:first-of-type`, `:nth-child(2n)...`
- son **état**, par exemple :
  - s'il s'agit d'un **lien** : `a:hover`, `a:active`, `a:visited...`
  - s'il s'agit d'un **élément de formulaire** : `input:focus`, `select:invalid`,  
`[ type="checkbox" ]:checked...`
- Il est également possible d'**exclure un élément** correspondant à un sélecteur grâce à la pseudo-classe `:not(<selecteur>)`

# Les sélecteurs

## Types de sélecteurs

Les **pseudo-éléments** permettent quand à eux :

- le **ciblage d'une sous-partie** d'un élément, par exemple :

`h1::first-letter`

`p::first-line...`

- l'**injection de contenu** :

- `::before` (avant le contenu de l'élément),
- `::after` (après le contenu de l'élément).

Il existe encore d'autres sélecteurs plus spécifiques, permettant un ciblage plus fin et la gestion de cas complexes : <https://css4-selectors.com/selectors/>

# Les propriétés

Le standard définit :

- à quels éléments **s'applique** chaque propriété,
- quelles sont les **types et valeurs possibles** de ces propriétés,
- quelles propriétés sont **héritées** ou non.

# Les propriétés

## Polices et textes

- `font-size` pour la **taille** de police :
  - Valeurs prédéfinies absolues (`medium`, `large`) ou relatives (`smaller`, `larger`)
  - Valeur numérique absolue en pixel (`px`), ou relative en `rem`.
- `font-weight` pour la **graisse** :
  - Valeurs prédéfinies : `light`, `lighter`, `normal`, `bold`, `bolder`
  - Valeur numérique entre 0 et 1000 (1000 étant le plus gras).
- `font-style` pour l'**inclinaison** :
  - Valeurs prédéfinies : `normal`, `italic`, `oblique`.
- `text-decoration` pour les effets de **décoration** (soulignage, surlignage) :
  - Valeurs prédéfinies : `underline`, `overline`, `line-through`, `none`

# Les propriétés

## Polices et textes

- `font-family` pour la **famille** :
  - Grandes familles : `serif` (avec empattements), `sans-serif` (sans empattements), `monospace` (à chasse fixe).
  - Nom de police, par exemple `Arial`, `Times New Roman`, `Helvetica`...
- `text-transform` pour les **majuscules et minuscules** :
  - Valeurs prédéfinies : `uppercase` (tout en majuscules), `lowercase` (tout en minuscules), `capitalize` (uniquement les premières lettres de chaque mot en majuscules)
- `font-variant`, qui prendra la valeur `small-caps` afin d'afficher des **petites capitales**.



# Les propriétés

## Couleurs

- `color` pour la **couleur de la police** :
  - Valeurs prédéfinies : `red`, `blue`, `green`, `orange`...  
Il en existe plus de 140 : [https://www.w3schools.com/cssref/css\\_colors.php](https://www.w3schools.com/cssref/css_colors.php)
  - Valeurs utilisateurs :
    - Hexadécimal : `#RRVVBB`,
    - RGB : `rgb(r, g, b)`, avec des valeurs allant de 0 à 255, ou de 0% à 100%,
    - RGB + opacité : `rgb(r, g, b, a)`
- `opacity` pour l'**opacité**, avec une valeur allant de 0 (transparent) à 1 (opaque).
- `background-color` pour la **couleur de fond**, acceptant les mêmes valeurs que `color`.

# Les propriétés

## Couleurs

Il est possible de créer des **fonds plus complexes** via la propriété générique **background**

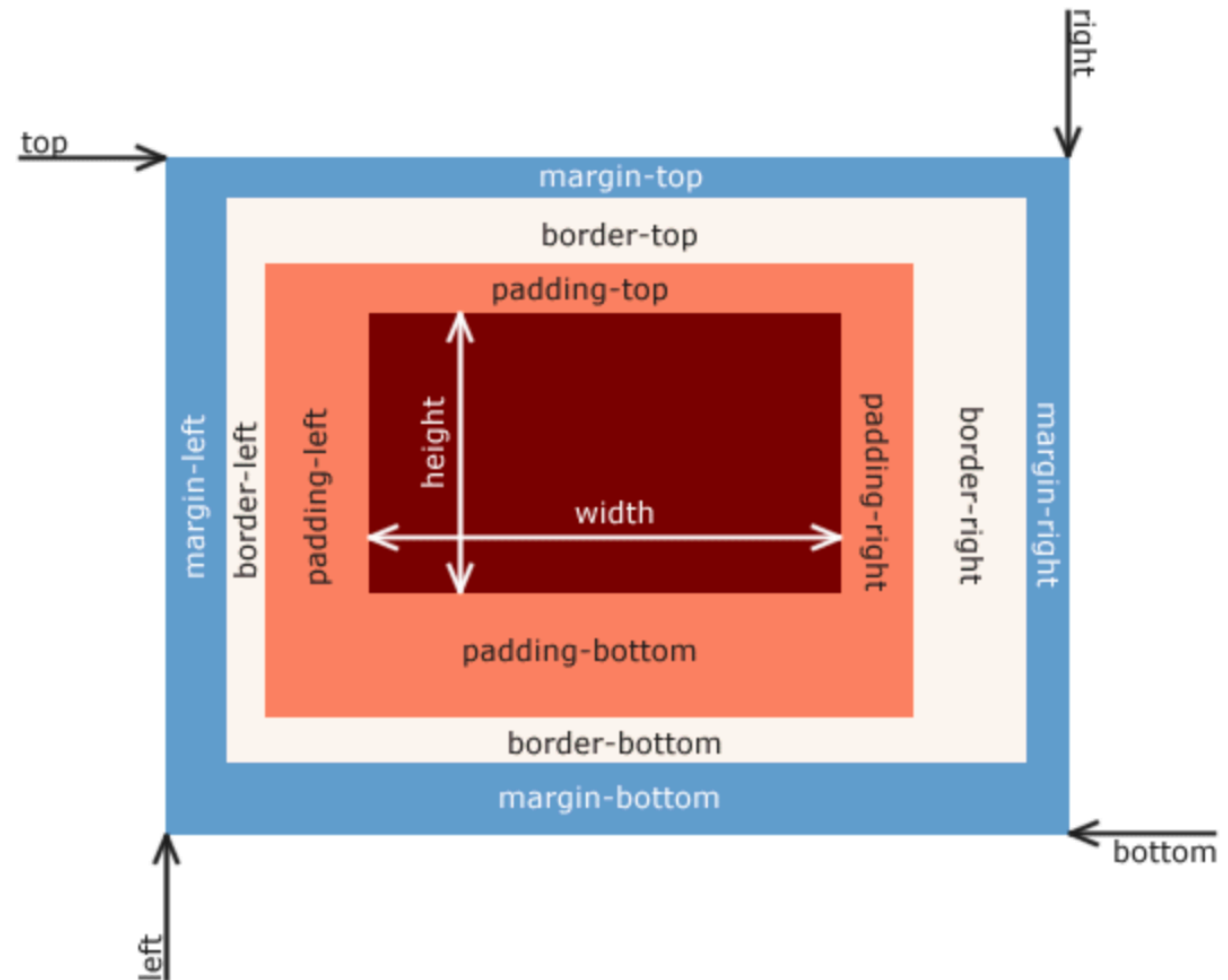
- Ajouter une image d'arrière-plan :  
<https://developer.mozilla.org/fr/docs/Web/CSS/background-image>
- Ajouter un gradient : <https://cssgradient.io/>

# Les propriétés

## Marges

Il en existe deux types :

- les **marges extérieures** ou margin, qui définissent la distance entre l'élément et ses voisins,
- les **marges intérieures** ou padding, qui définissent la distance entre les descendants de l'élément et ses bordures.



# Les propriétés

## Marges

On constate sur ce schéma représentant le **modèle de boîte standard** que celui-ci ne prend pas en compte les marges intérieures et les bordures dans le calcul des dimensions (`width` et `height`).

Il est cependant possible d'utiliser un **modèle alternatif** qui les incluent.

Afin de l'appliquer à l'intégralité des éléments de vos documents, il suffit d'ajouter ce code au début de votre feuille de style :

```
html {  
    box-sizing: border-box;  
}  
*,  
*::before,  
*::after {  
    box-sizing: inherit;  
}
```

# Les propriétés

## Marges

Attention à prendre en compte l'effet de **fusion des marges** : si les marges haute (`margin-top`) et basse (`margin-bottom`) de deux éléments visuellement adjacents et ayant le même ancêtre se touchent, elles peuvent dans certains cas fusionner, en prenant la valeur de la plus grande des deux.

# Les propriétés

## Bordures

La propriété `border` permet de définir dans l'ordre l'épaisseur, le style et la couleur de la **bordure** d'un élément, par exemple :

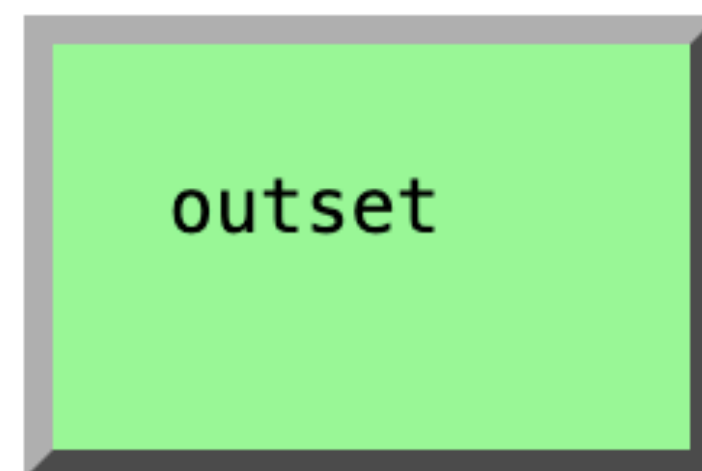
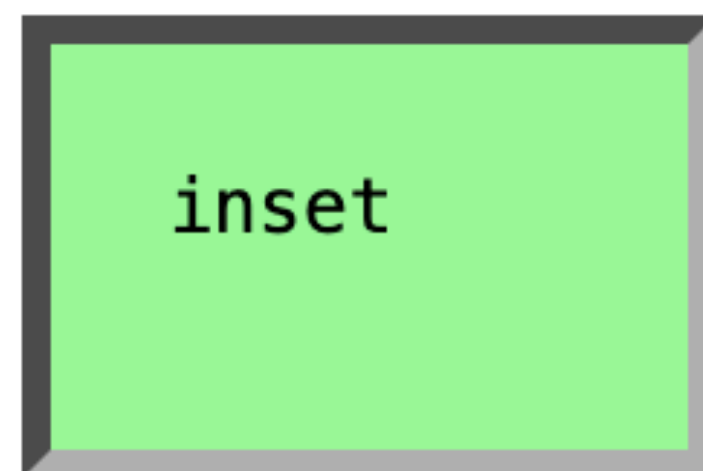
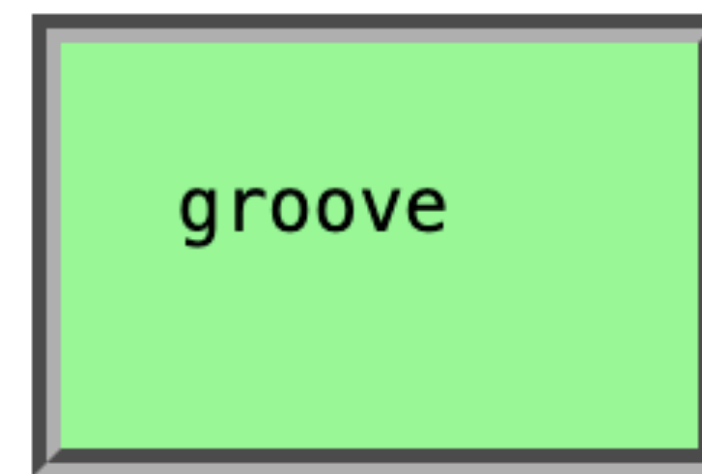
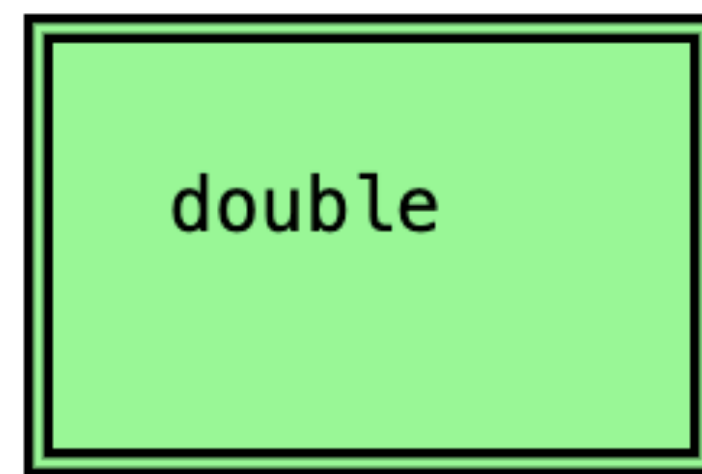
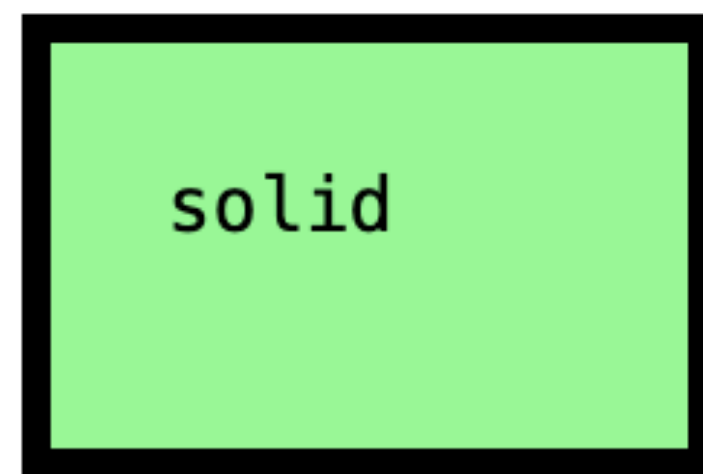
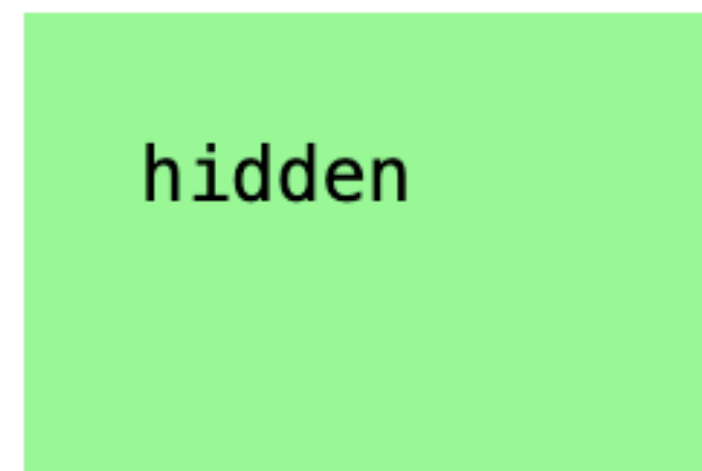
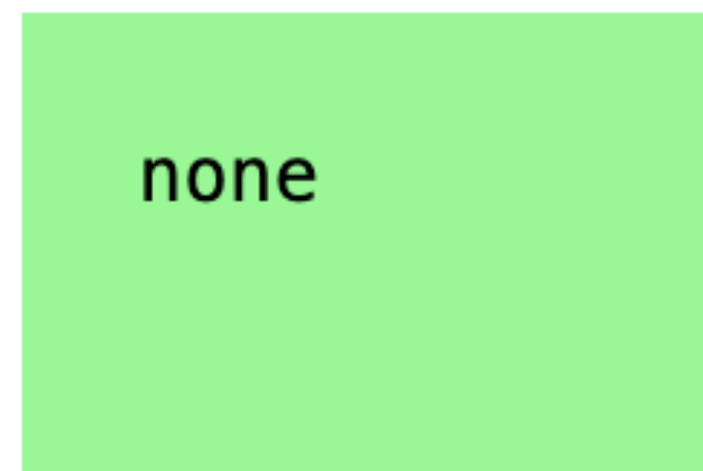
```
border: 5px dotted red
```

L'**épaisseur** est généralement indiquée en pixel, et la **couleur** peut prendre une valeur de même type que les polices ou les fonds.

Le **style** peut prendre les valeurs prédéfinies présentés au slide suivant.

# Les propriétés

Bordures > Styles prédéfinis





# Commentaires

Tout comme en HTML, il est possible d'ajouter des commentaires pour organiser son code :

```
/* un commentaire sur une règle ou un ensemble de règles */  
.element {  
    /* un commentaire spécifique sur une propriété */  
    propriete: valeur; /* on peut le mettre sur la même ligne */  
}
```

# Sites web réactifs

De nos jours, la consultation d'un site web peut se faire sur des appareils très différents, par exemple : ordinateur de bureau, ordinateur portable, smartphone, téléviseur, montre connectée...

Pour permettre un affichage optimal sur tous ces supports, il est possible de recourir à différents **principes de conception de sites réactifs** - aussi appelé "**responsive web design**".

# Sites web réactifs

## Media queries

Cette technique permet l'**application de styles différents** en fonction du **contexte** :

- taille d'écran,
- navigateur de l'utilisateur,
- version imprimable,
- ...

# Sites web réactifs

## Media queries

Elle repose sur l'utilisation de **règles conditionnelles** au sein d'une feuille de styles, via la requête `@media`

```
main {  
    width: 1200px;  
    padding: 20px;  
}  
  
@media only screen and (max-width: 720px) {  
    main {  
        width: 100%;  
        padding: 10px;  
    }  
}
```

# Sites web réactifs

## Media queries

La syntaxe de ces requêtes repose sur des **opérateurs logiques** :

- **and** : les styles sont appliqués si toutes les conditions sont tenues,
- **,** : les styles sont appliqués si au moins une des conditions est tenue,
- **not** : les styles ne doivent *pas* être appliqués si la condition est rencontrée,
- **only** : les styles ne sont appliqués *que* si la condition est rencontrée.

# Sites web réactifs

## Media queries

Le type de média est optionnel, et peut prendre les valeur suivants :

- **screen** : styles destinés aux écrans,
- **print** : styles destinés à l'impression et aux documents paginés (ex. PDF),
- **all** : valeur par défaut, correspondant à tous les types.

# Sites web réactifs

## Media queries

Enfin, il existe un certain nombre de caractéristiques de médias, par exemple :

- `min-width` / `max-width` : sans doute la plus utilisée, elle permet d'appliquer des styles **en fonction de la largeur de l'écran**, et sa valeur est généralement exprimée en `px`, mais il est possible aussi d'utiliser des `%` ou l'unité `vw` - viewport width, qui va de 0 à 100,
- `min-height` / `max-height` : même principe, mais **en fonction de la hauteur d'écran**,
- `orientation` : permet d'appliquer des styles **en fonction de l'orientation de l'appareil**, ce qui peut être utile sur mobile et tablette. Elle peut prendre les valeurs `portrait` ou `landscape` (paysage).



# Sites web réactifs

## Media queries

D'autres caractéristiques correspondent à des **réglages utilisateurs**, et peuvent aider à améliorer l'expérience utilisateur et l'accessibilité du site, par exemple :

- `prefers-reduced-motion` qui indique la préférence de l'utilisateur pour une limitation des animations sur la page,
- `prefers-color-scheme` qui permet d'ajuster automatiquement le thème du site en fonction de la préférence de l'utilisateur pour un thème clair (`light`) ou sombre (`dark`).

# Sites web réactifs

## Flexbox

**CSS Flexible Box Layout** est un modèle de mise en page sur **une seule dimension**, c'est-à-dire que l'organisation des éléments se fait dans une direction à la fois - ligne ou colonne - mais pas les deux ensembles.

Afin de transformer un élément HTML en **conteneur flexible**, on l'indique dans sa propriété `display` :

```
.container {  
    display: flex; /* ou inline-flex */  
}
```

# Sites web réactifs

## Flexbox

Propriétés des conteneurs flex :

- `flex-direction` permet d'indiquer la direction de l'**axe principal** sur lequel les descendants seront organisés. Elle peut prendre les valeurs `row` (par défaut), `row-reverse`, `column`, `column-reverse`.
- `flex-wrap` permet, lorsqu'elle prend la valeur `wrap`, de créer un conteneur flexible **sur plusieurs lignes**.  
A contrario, la valeur `nowrap` force les éléments à se redimensionner en empêchant le passage à la ligne - s'il n'y a pas assez d'espace disponible, cela causera un dépassement.

# Sites web réactifs

## Flexbox

Propriétés des conteneurs flex (suite) :

- `justify-content` définit l'**alignement** des descendants directs suivant l'**axe principal**, et peut prendre les valeurs `flex-start`, `flex-end`, `center`, `space-between`, `space-around`, `space-evenly`, `stretch`...
- `align-items` définit l'alignement des éléments suivant l'**axe transversal**, qui est toujours perpendiculaire à l'axe principal et peut prendre les valeurs `flex-start`, `flex-end`, `center`, `stretch`...
- `align-self` a un comportement comparable à `align-items` (et accepte les mêmes valeurs), mais il ne s'applique que sur les éléments ciblés par le sélecteur CSS, si leur parent est un conteneur flex.

# Sites web réactifs

## Flexbox

Attention, **ces propriétés ne s'héritent pas** : elle ne s'applique qu'à l'élément cible et/ou à ses descendants.

Toutes ces explications étant un peu arides, un développeur a décidé de créer une application permettant d'expérimenter visuellement les effets des différentes valeurs des propriétés des éléments Flex : Flexbox Froggy - <https://flexboxfroggy.com/#fr>

En bonus, un petit pense-bête pour éviter d'avoir à retenir toutes les propriétés ainsi que leurs valeurs possibles d'un seul coup : <https://github.com/alsacreations/kiwipedia/blob/main/resources/flexbox-cheatsheet.png>

# Sites web réactifs

Pour aller plus loin...

- **CSS Grid Layout** est un autre modèle de mise en page via la propriété `display`, cette fois-ci **sur deux dimensions**.  
Le site interactif Grid Garden <https://cssgridgarden.com/#fr> permet d'en comprendre les ressorts.  
Pense-bête :  
<https://github.com/alsacreations/kiwipedia/blob/main/resources/grid-cheatsheet.png>
- Les **Container Queries** sont basés sur les mêmes principes que les Media Queries, mais au lieu de prendre comme référence les dimensions de la fenêtre, ils interrogent les dimensions de l'ancêtre d'un élément pour déterminer si des styles doivent être appliqués.  
Ressource en français :  
<https://www.alsacreations.com/article/lire/1915-Les-Container-Queries-en-CSS.html>