Conception Et Programmation Web

LIFWEB - http://lifweb.pages.univ-lyon1.fr/

Semestre printemps 2024-2025
L3 - UCBL

Aurélien Tabard - https://tabard.fr/, basé sur les cours de Romuald Thion

HTML - CSS - DOM

Aurélien Tabard - 2024-2025 - basé sur les cours de Romuald Thion

HTML - CSS - DOM

Plan

1. Rappels : composition d’une page Web
2. Le Document Object Model
3. Abstractions au dessus du DOM
4. Événements
5. Chargement et exécution du JavaScript
6. Bilan et TP

HTML - CSS - DOM

Plan

1. Rappels : composition d’une page Web
2. Le Document Object Model
3. Abstractions au dessus du DOM
4. Événements
5. Chargement et exécution du JavaScript
6. Bilan et TP

Les trois langages du Web

HTML5, CSS3, JavaScript et HTTP!

Éléments d’une page web

Contenu - HTML

Structure - HTML

Style - CSS

Comportement - JavaScript

HTML

HyperText Markup Language

Langage de balises utilisé pour structurer le contenu d'un document destiné à être affiché par un navigateur.

Liste de 100+ balises : http://html5doctor.com/

À chaque balise HTML est associé une sémantique.

Chaque élément de contenu est placé dans une balise.

L'imbrication des balises donne une structure d'arbre.

Anatomie d’une page Web

Une (bonne) page HTML commence par un DOCTYPE

Ensuite <html> suivi de <head> et de <body>

Les méta-informations vont dans <head>

Le contenu va dans <body>

```
<!DOCTYPE html>
<html lang="en">
    <head>
        ...meta stuff...
    </head>
    <body>
        ...your website...
    </body>
</html>
```

<!DOCTYPE html>

Doctype

- Permet de dire au navigateur comment lire le HTML, et de construire l'arbre associé.
- Plusieurs versions de HTML différentes, le doctype permet de les différentier (c.f. DTD avec XML)

- Au début du document
- Doit correspondre à la version HTML utilisée
- Des validateurs existent pour vérifier que le HTML est correct et correspond au doctype déclaré
- Aujourd'hui, avec HTML5 : <!DOCTYPE html>

Contenu de <head>

Titre du document <title></title>

Autre informations non affichées à l'écran, utilisées par le navigateur, les moteurs, etc.

- <meta name="..." content="..." />
- <meta http-equiv="Refresh" content="4" ; url="http://www.google.com" />
- <meta name="author" content=""/>
- <meta name="Keywords" content="motcle1, motcle2, motcle3"/>
- <meta name="language" content="fr"/>
- URL de base pour les URL relatives <base href="URL-de-base" />

Styles

- <style /> : inclure une feuille de style CSS dans la page
- <link /> lier le document à une ressource externe (typiquement, feuille de style)

Scripts

- <script /> ajouter un script à la page

Le contenu de <body>

le corps de la page

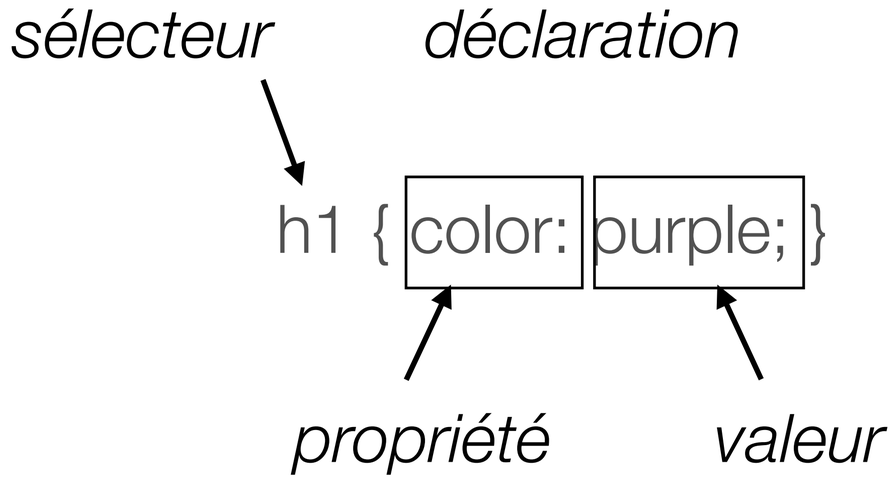
Elément <body>

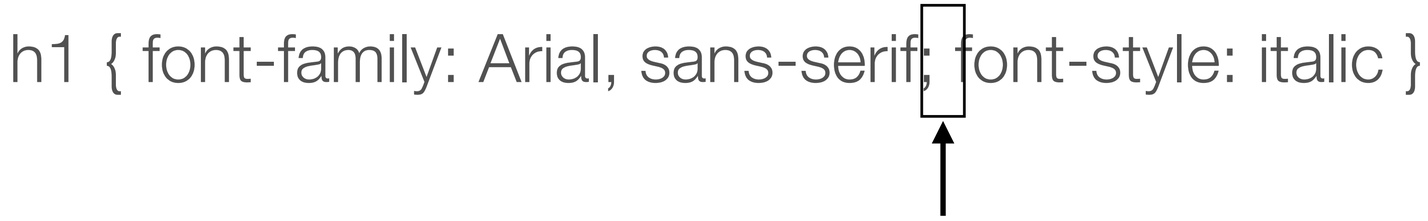
- Le contenu du site

Deux grandes classes d’éléments :

- Block
  - Prennent toute la largeur disponible
  - Provoque un retour à la ligne
  - Peuvent contenir des éléments inline
- Inline
  - Ne provoque pas de retour à la ligne
  - Ne peuvent pas contenir des éléments block
  - Peuvent contenir des éléments inline

Structure des règles





Exemple

```
body {
        background: #FFFFFF;
        color: black; /* commentaire */
        margin-left: 5%;
        margin-right: 5%;
        font-family: Tahoma, Optima,
        Arial, sans-serif;
}
```

Types de sélecteur

- Simples et groupes
- Classes
- Pseudo-classes
- Pseudo-éléments
- Contextuels

Sélecteurs CSS : simples et groupes

Simple

- Lié à un type d'élément HTML
- Utilisation de son nom
- Exemple : `h1 { text-align: center; }`

Groupe

- Regroupement de règles qui s'appliquent à plusieurs éléments
- Exemple : `h2, p { font-family: Optima, Arial, sans-serif; }`

Impossibilité de considérer différemment des éléments de même type

Sélecteurs CSS : classes

- On peut assigner une classe à un élément HTML
  `<h1 class="centre" >`
- Celle-ci spécifie un sélecteur particulier dans le feuille de style
  `h1.centre { text-align: center; }`
- Une classe peut s'appliquer à de multiples éléments
  `.centre { text-align: center; }`
- s'appliquera aussi à <h2 class="centre">, etc.

Sélecteur CSS : pseudo-classes

- Sélecteurs qui sélectionnent des éléments en fonction de leur état à un moment donné
- Exemple et intérêt principal
  - **a:link** — lien non visité et inactif
  - **a:hover** — lien sur lequel passe le pointeur de la souris
  - **a:active** — lien sur lequel on clique
  - **a:visited** — lien déjà visité
- Exemple
  - `a:link {color: blue;}`
  - `a:visited {color: magenta;}`
  - `a:hover {  color: red; text-decoration:none; font-weight: bold;}`
  - `a:active {color: red;}`

Sélecteurs CSS : pseudo-éléments

:first-letter

- première lettre dans un élément bloc (ex. p, h1, …)

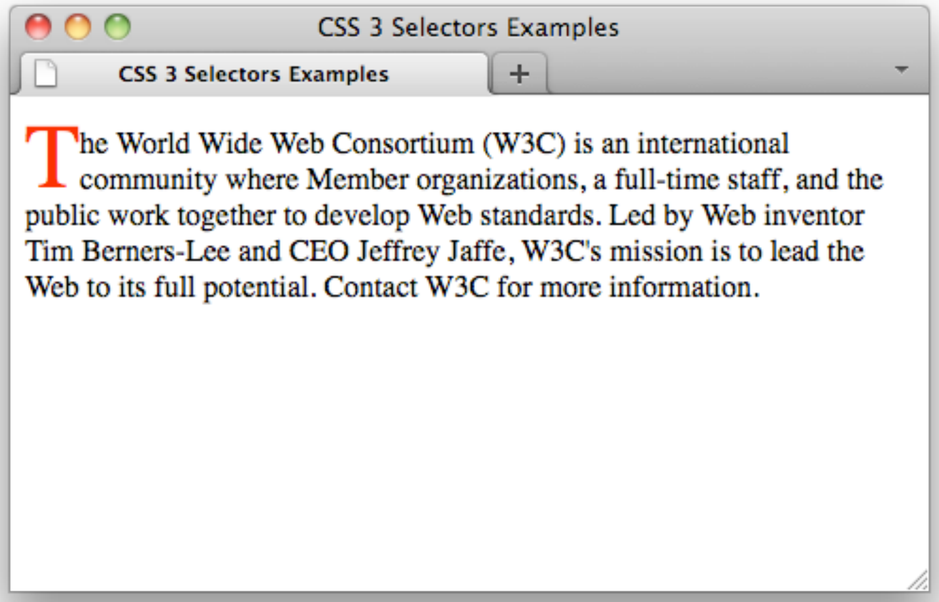
:first-line

- première ligne dans un élément bloc (ex. p, h1, ...)

:after

:before

- avant et après du contenu



http://www.w3.org/wiki/CSS3/Selectors#Pseudo-elements

Sélecteurs CSS : contextuels

Sélecteurs qui ne sélectionnent que des éléments dans un certain contexte

- Les plus complexes mais aussi les plus riches.

Exemple de style contextuel :

CSS :

```
h1 em { color: red; }
```

HTML :

```
<h1>Ceci est un texte de header <em>ce
texte est mis en évidence</em> celui-ci ne
l'est pas.</h1>
  <p>Dans ce paragraphe,
  <em>ceci est mis en évidence</em>
</p>
```

Propriétés de texte

- font-size: small | medium...| % | x pt
- font-family: fontname1, fontname2 *(si la première n'est pas disponible)*, familyname (serif, sans-serif, etc.)
- font-weight: bold | lighter
- font-style: italic, oblique
- text-align: left | center | right | justify
- text-indent: *(retrait de première ligne)* % | 2 px

Les couleurs

color:

- red | blue...| hexcode

background-color:

- red | blue...| hexcode

Hex Code	Color	Hex Code	Color	Hex Code	Color
#FFFFFF		#CCFFFF		#99FFFF	
#FFFFCC		#CCFFCC		#99FFCC	
#FFFF99		#CCFF99		#99FF99	
#FFFF66		#CCFF66		#99FF66	
#FFCCFF		#CCCCFF		#99CCFF	
#FFCCCC		#CCCCCC		#99CCCC	
#FFCC99		#CCCC99		#99CC99	
#FFCC66		#CCCC66		#99CC66	
#FF99FF		#CC99FF		#9999FF	
#FF99CC		#CC99CC		#9999CC	

Hex Code	Color	Hex Code	Color	Hex Code	Color
#FF9999		#CC9999		#999999	
#FF9966		#CC9966		#999966	
#FF66FF		#CC66FF		#9966FF	
#FF66CC		#CC66CC		#9966CC	
#FF6699		#CC6699		#996699	
#FF6666		#CC6666		#996666	
#FF6633		#CC6633		#996633	
#FF6600		#CC6600		#996600	
#FF33FF		#CC33FF		#9933FF	
#FF33CC		#CC33CC		#9933CC	

Types d’éléments

Éléments blocs

- Prennent la forme d’un bloc dans la page
  - ensemble de lignes
  - ne peuvent être contenus que dans d’autres éléments blocs
- Exemple : p, img, ul, table, h1, div...

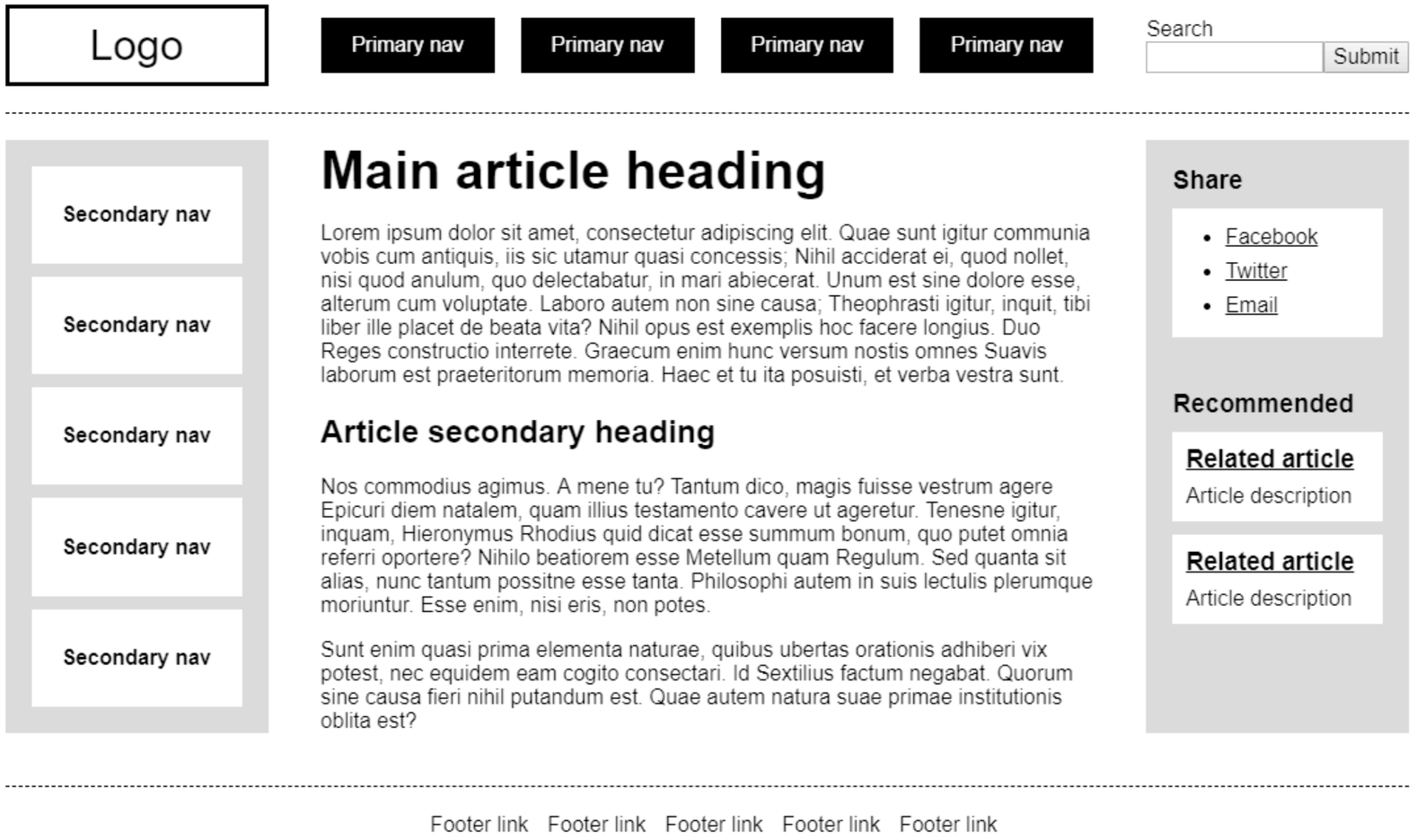
Éléments inline

- S’inscrivent dans la continuité des éléments
  - ne forcent pas un changement de ligne
  - peuvent être inclus dans n’importe quel élément
- Exemple : a, em, span…

Éléments de listes

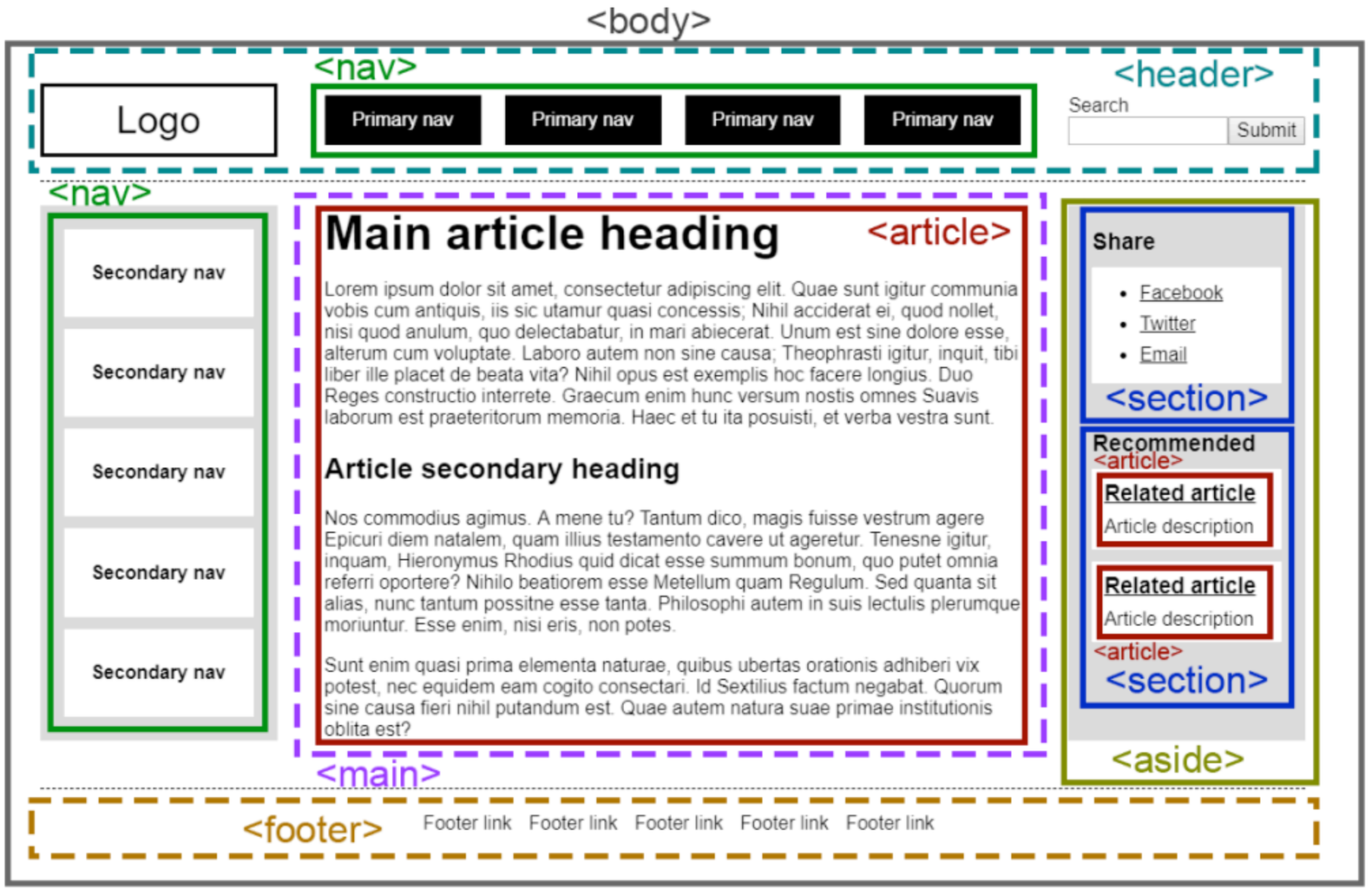
- éléments HTML qui ont un marqueur (ex: bullet) et un ordre

CSS



Certaines balises sont liés à la structure logique de la page : https://css-tricks.com/how-to-section-your-html/

CSS



Certaines balises sont liés à la structure logique de la page : https://css-tricks.com/how-to-section-your-html/

Propriétés graphiques : visibilité

Attribut visibility et display

- indiquent si (visible|hidden) et comment un élément est affiché

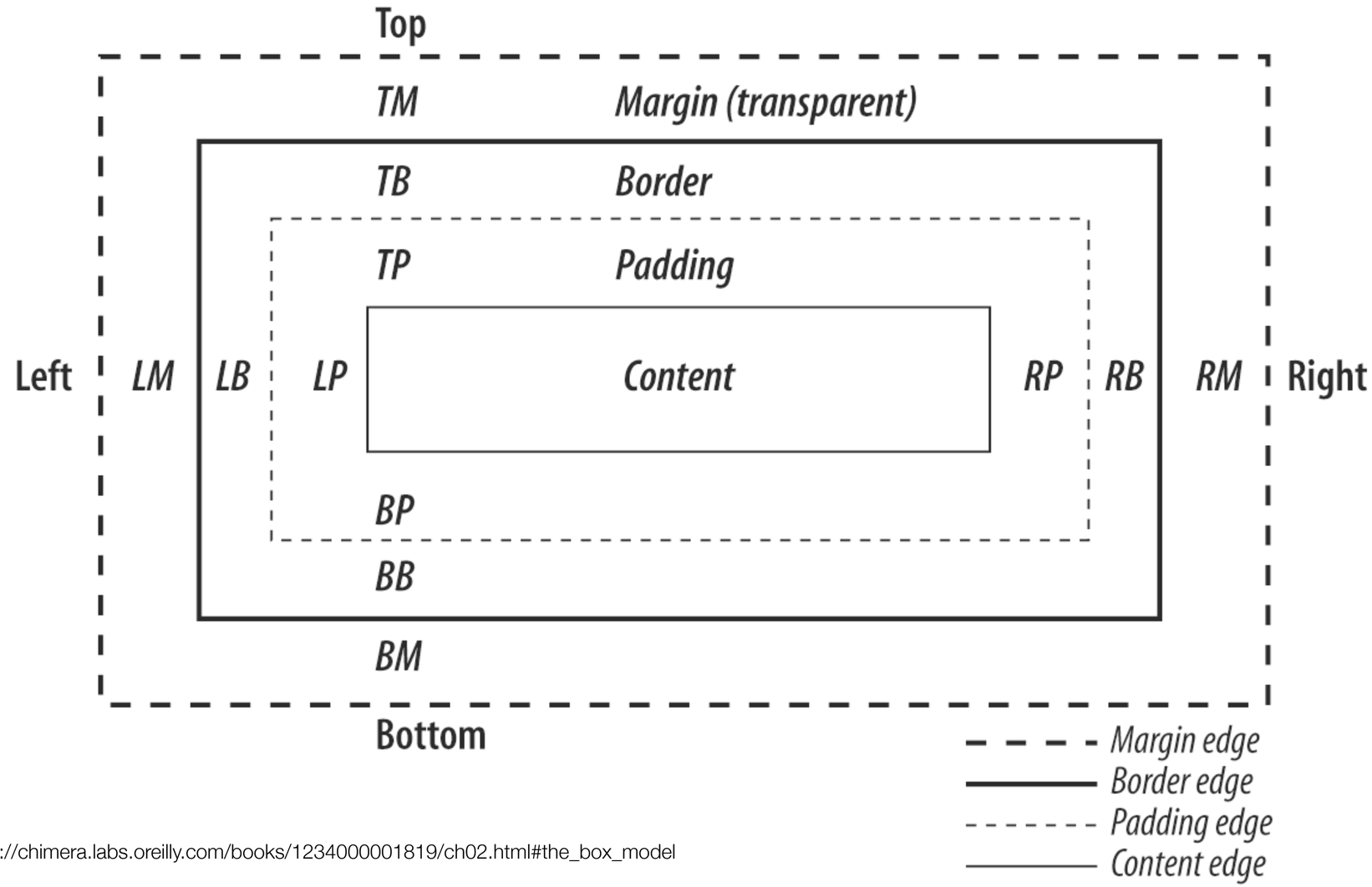
Attribut z-index

- en cas de superposition de blocs d'affichage, indique l'ordre dans lesquels le navigateur doit les afficher (cf. couches/layers dans les logiciels de dessin)

Remarques

- les distances s'expriment en pixels (px), points (pt), unités métriques (cm, mm), ou pourcentages de la taille de la fenêtre (%) ou en unité relative (em, rem)

Le modèle de boite (box model)



http://chimera.labs.oreilly.com/books/1234000001819/ch02.html#the_box_model

Héritage de style

- Par défaut
  - Les styles d'un élément sont héritées par ses éléments descendants
  - Exemple :

```
<p style="color: red;"> Du texte <em>mis en évidence<em>,
pas mis en évidence.<p>
```

Du texte *mis en évidence*, pas mis en évidence.

- A condition que l'héritage ait un sens
  - i.e. que les caractéristiques soient applicables à l'élément enfant
    - un positionnement de bloc n'a pas d'intérêt pour un élément em qui y est contenu
  - si un style est défini spécialement pour un élément
  - ex: `em { color: blue; }`, l'héritage ne se fait pas

Du texte *mis en évidence*, pas mis en évidence.

Cascading style sheets : cascade

On peut avoir concurrence entre plusieurs styles définis dans de multiples endroits

- styles par défaut (1- navigateur)
- fichiers CSS externes (2- spécifications globales au site)
- élément head du document XHTML (3- spécification globales au doc.)
- attributs style des éléments (4- spécification locales)
- style utilisateur (5- spécification de l’utilisateur)

Cascading style sheets : priorité

Notion de cascade ou ordre de priorité des styles

- trouver toutes les déclarations qui s'appliquent à un élément
- les classer par spécificité
  - système de poids qui s'ajoutent
- les classer par ordre d'apparence
  - plus une déclaration apparaît tard, plus elle a de poids

Exemple

- style (4) > style (4 hérité) > style (3) > style (2) > style (1)

HTML - CSS - DOM

Plan

1. Rappels : composition d’une page Web
2. Le Document Object Model
3. Abstractions au dessus du DOM
4. Événements
5. Chargement et exécution du JavaScript
6. Bilan et TP

HTML génère un arbre

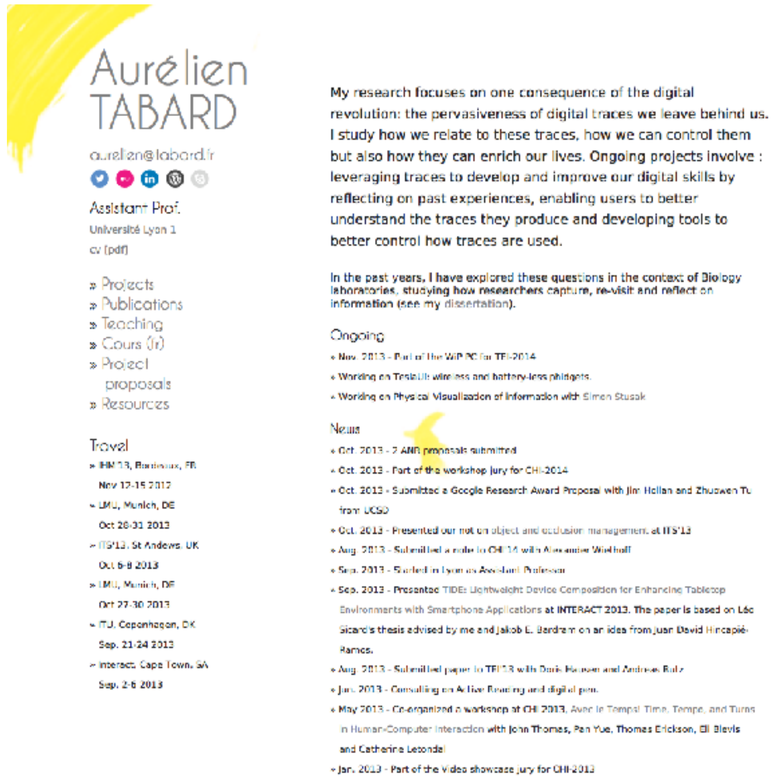
Le navigateur affiche les documents HTML (+ CSS et javascript)

- Demander (GET) et recevoir le HTML du serveur en document text/html
- Parser le document et gérer les erreurs
- Interpréter le document comme s'il n'avait pas d'erreur
- Recevoir toutes les resources (CSS, images, JavaScript, …)
- Construire un modèle interne (DOM)
- Appliquer les règles CSS qui définissent le style du document document (e.g., marges and taille du texte)
- Afficher visuellement la structure et le contenu
- Commencer à exécuter le code Javascript
- Répondre aux évènements (clavier, souris, timer) et exécuter le code.

Plus sur CSS par la suite

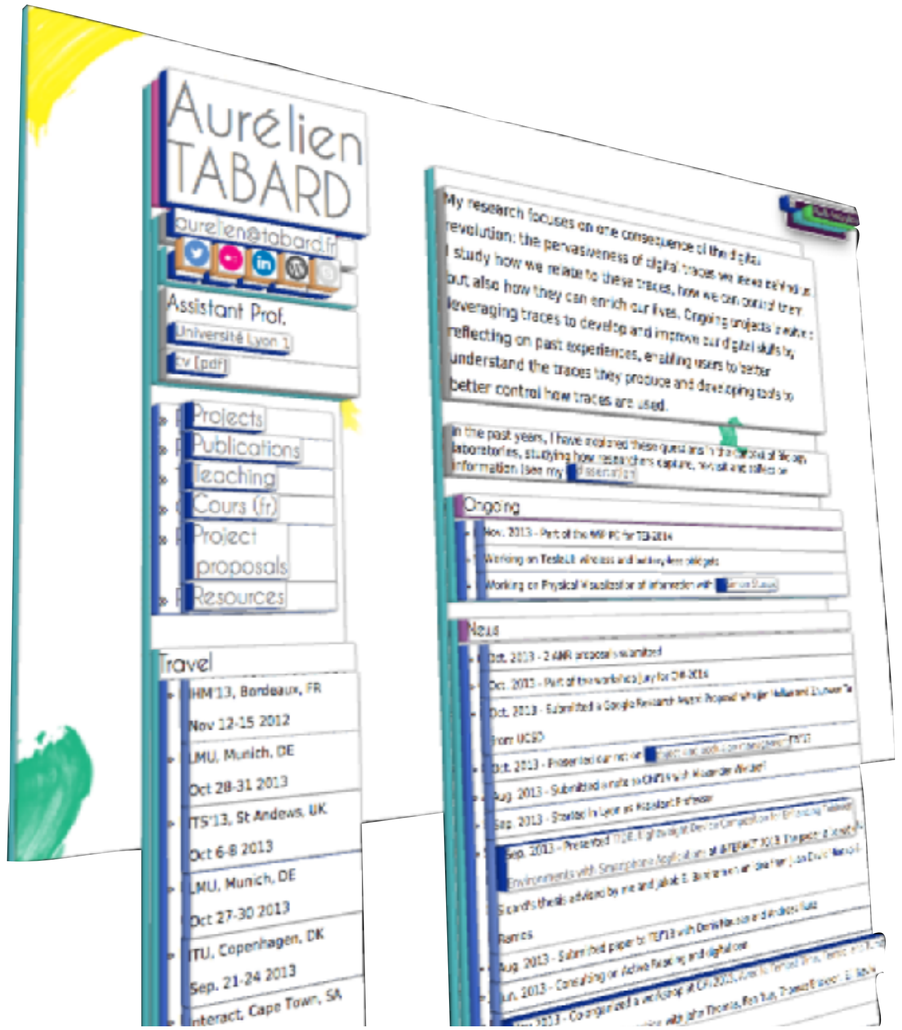
HTML génère un arbre

- Un arbre contient une racine,
  des parents, des frères, des enfants
- comme un arbre généalogique



HTML génère un arbre

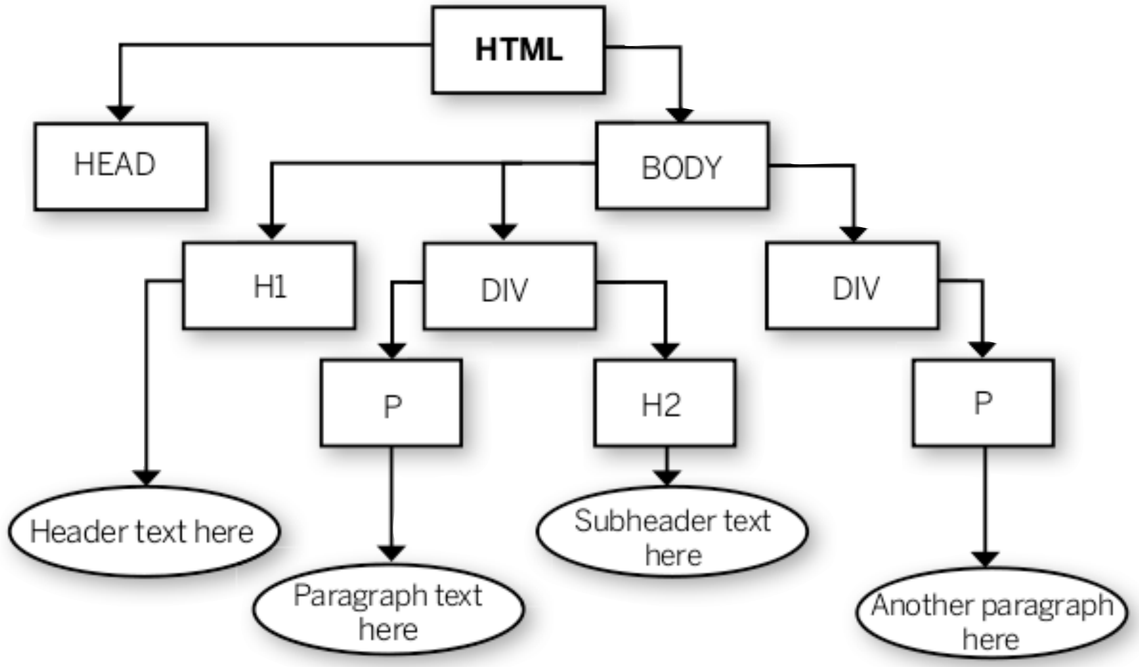
- Un arbre contient une racine,
  des parents, des frères, des enfants
- comme un arbre généalogique



HTML génère un arbre

https://javascript.info/dom-nodes

- Un arbre contient une racine,
  des parents, des frères, des enfants
- comme un arbre généalogique



Le DOM

Document Object Model

Modèle sous forme de structure arborescente d'un document (page) Web

Accessible via une API du navigateur

Cette API (interface) standard permet d'interagir programmatiquement avec le navigateur :

- documents HTML, XML, SVG, MathML, CSS, etc.
- navigateur, onglets, etc.

Spécification DOM (spécification, Wikipédia)

Références sur le DOM

javascript.info - Browser: Document, Events, Interfaces

MDN - Document Object Model (DOM)

MDN - Examples of […] development using the DOM

Le DOM est une API

Tous les navigateurs proposent une interface JS
avec une implémentation du DOM :

- L'API DOM est aussi disponible dans d'autres langages,
  sous forme de bibliothèque (tierce), e.g., xml.dom en Python.
- Sans le DOM et les autres APIs, JS ne ferait rien

☢️ JS s'exécute dans un bac à sable par sécurité ☢️

Autre API des navigateurs

Les navigateurs proposent de nombreuses Web APIs autres que le DOM :

- Pour CSS, voir par exemple CSSOM
- Window et Navigator
- Fetch API ou WebSocket API
- …

La racine de l’arbre est globalThis.document

Les éléments HTML sont des Node de l’arbre

- Difference between Node and Element?

Tout est représenté dans l’arbre

- Y compris commentaires et scripts !
- Voir MDN - Node:nodeType property

L’arbre DOM est live :

- les modifications effectuées en JS sont immédiatement visibles dans le navigateur ;
- les modifications effectuées dans le navigateur via les DevTools sont visibles en JS.

Exemple d’arbre DOM live

Sur une page initialement vide :

```
const $body = document.body; // HTMLBodyElement

console.debug($body.childNodes);
// NodeList []

$body.innerHTML = "<h1>Titre</h1><p>Bonjour</p>";

console.debug($body.childNodes);
//NodeList [ h1, p ]

$body.innerHTML = "";

console.debug($body.childNodes);
// NodeList []
```

Principales interfaces de l'API DOM

Interface	Description
`Document`	La classe de la racine de l'arbre DOM, l'objet `document`.
`EventTarget`	La classe la plus abstraite, la gestion événementielle.
`Node`	Un nœud abstrait de l'arbre, soit un `Element`, soit d'un autre type.
`Element`	Une interface abstraite pour les éléments, hérite de `Node`, commun à `HTMLElement` et `SVGElement`.
`HTMLElement`	Un élément HTML est spécialisé par les éléments concrets comme `HTMLAnchorElement`
`NodeList`	Un tableau de nœuds, pour stocker les fils par exemple. Similaire, mais *non identique* à `Array` JS.
	…

Hiérarchie d’interfaces du DOM des éléments anchors

[MDN - HTMLAnchorElement]

Sélection des éléments

`Element.querySelector(selector)` sélectionne le premier élément trouvé

`Element.querySelectorAll(selector)` sélectionne tous les éléments

autres méthodes, *à éviter*

- ~~`globalThis.id` ⚠️~~
- ~~`Element.getElementById(id)`~~
- ~~`Eement.getElementByClassName(id)`~~
- ~~`Element.getElementByName(name)`~~
- …

Sélecteurs

Les sélecteurs sont des strings dans la syntaxe des sélecteurs CSS.

```
document.querySelectorAll("#box li > a:visited");
```

Voir javascript.info *searching DOM elements*

Résultat d’un querySelector

Propriété `Node.childNodes` (MDN) → renvoie une `NodeList` live.

Propriété `Element.childNodes` (MDN) → renvoie une `HTMLCollection` live.

📔 Il est conseillé de transformer ces collections avec `Array.from(iterable)` (MDN - Array.from) → voir MDN - HTMLCollection

What is the difference between children and childNodes in JavaScript?

Création et ajout d’éléments

`document.createElement(tag, options)` crée un élément HTML (MDN)

- ⚠️ l’élément est créé, mais est invisible

pour ajouter l’élément à l’arbre DOM :

- 👎 `Node.appendChild(aNode)` ajoute un dernier fils à un Node (MDN)
- 👍 `Element.append(nodeOrStr1, ..., nodeOrStrN)`
  similaire pour la classe Element (MDN) (unicorn)

Manipuler les éléments du DOM

L'API DOM est complète : de nombreuses méthodes pour le remplacement, la suppression, etc. :

- `Node.cloneNode(deep)` MDN
- `Node.replaceChild(newN, oldN)` MDN
- `Element.replaceWith(e1, …, eN)` MDN
- `Element.replaceChildren(e1, …, eN)` MDN
- `Element.remove()` MDN
- …

Manipulation des attributs

`Element.getAttribute(name)` MDN

`Element.setAttribute(name, val)` MDN

`Element.removeAttribute(name)` MDN

Exemple :

```
document.querySelector("img").getAttributeNames();
// Array [ "src", "alt" ]
document.querySelector("img").getAttribute("src");
// "../img/javascriptinfo_html.png"
document.querySelector("img").src;
// "http://127.0.0.1:8080/img/javascriptinfo_html.png" (FQDN)
```

voir MDN - HTMLImageElement.src

Enfant et contenu

⚠️ `Element.innerHTML` MDN

- pour lire ou écrire un contenu HTML
- à éviter pour performance et sécurité

`HTMLElement.innerText` MDN

👍 `Node.textContent` MDN

⚠️ `innerText` et `textContent` suppriment tout le contenu. ⚠️

On préfère textContent à innerText (et à innerHTML)

- unicorn
- SO - What's the use of textContent/innerText when innerHTML does the job better?
- SO - Difference between textContent vs innerText).

Danger de Element.innerHTML

Les failles de type Cross Site Scripting (XSS)

- OWASP - XSS
- OWASP - DOM Based XSS Prevention

```
const $div = document.createElement("div");
$div.innerHTML = `<img src="!" onerror="alert('XSS')"> `;
document.body.append($div);
```

☣ *L'ajout **non contrôlé** de contenu en provenance de l'utilisateur conduit à des failles (OWASP Top 10).* ☣

Ajouter des images

exemple

```
const cats = [
  { id: "k3", url: "https://cdn2.thecatapi.com/images/k3.jpg" },
  /* … */
  { id: "d4d", url: "https://cdn2.thecatapi.com/images/d4d.jpg" },
];

const $catsContainer = document.querySelector("#cats-container");
for (const cat of cats) {
  const $img = document.createElement("img");
  $img.setAttribute("src", cat.url);
  $img.setAttribute("alt", `Cat ${cat.id}`);
  $catsContainer.appendChild($img);
}
```

Ajouter des images

exemple

Variante avec fragment HTML pour ne rafraichir la page qu'une fois.

```
const $root = document.createDocumentFragment();
const $catsContainer = document.querySelector("#cats-container");
for (const cat of cats) {
  const $img = document.createElement("img");
  $img.setAttribute("src", cat.url);
  $img.setAttribute("alt", `Cat ${cat.id}`);
  $root.append($img);
}
$catsContainer.append($root);
```

HTML - CSS - DOM

Plan

1. Rappels : composition d’une page Web
2. Le Document Object Model
3. Abstractions au dessus du DOM
4. Événements
5. Chargement et exécution du JavaScript
6. Bilan et TP

Alternatives au DOM explicite

L’API DOM est complexe, à cause (notamment) de 20 ans d’évolution des standards avec maintien de la rétro compatibilité par les navigateurs.

- Avoir de bonnes pratiques.
- Outiller le développement.
- Utiliser des objets/outils de plus haut niveau.

La manipulation explicite du DOM est fastidieuse 🥺 Des solutions permettent de l’éviter.

Templating

= utilisation de modèles

On peut utiliser des moteurs de templating plus ou moins riches pour la création d'éléments complexes :

- https://mustache.github.io/
- https://handlebarsjs.com/
- https://ejs.co/
- https://pugjs.org
- https://jinja.palletsprojects.com (Python)

On utilisera côté serveur, mais pas côté client.

Exemple de mustache.js

```
<html>
  <body onload="renderHello()">
    <div id="target">Loading...</div>
    <script id="template" type="x-tmpl-mustache">
      <p> Hello {{ name }}! </p>
    </script>

    <script src="https://unpkg.com/mustache@latest"></script>
    <script src="render.js"></script>
  </body>
</html>
```

```
function renderHello() {
  const template = document.getElementById("template").innerHTML;
  const rendered = Mustache.render(template, { name: "Luke" });
  document.getElementById("target").innerHTML = rendered;
}
```

Frameworks clients

pour le master

Les framework s’appuient sur des composants qui encapsulent
contenu (HTML), présentation (CSS) et comportement (JS)
via des composants standardisés (Web Components) ou propres (React DOM) :

- https://react.dev/ ou https://preactjs.com/
- https://angular.io/
- https://vuejs.org/

Exemple React en JSX

JSX = JS + template HTML

```
// fichier JSX
const Index = (name) => {
  return <div>Hello {name}</div>;
};

// un élément Index ajouté à root
ReactDOM.render(<Index />, root);
```

```
// fichier JS obtenu après transpilation
const Index = (name) => {
  return React.createElement("div", null, "Hello ", name);
};
ReactDOM.render(React.createElement(Index, null), root);
```

“Web Components is a suite of different technologies allowing you to create reusable custom elements — with their functionality encapsulated away from the rest of your code — and utilize them in your web apps.”

MDN | Web Components

Web Components

Combinaison de trois technologies standardisées (javascript.info) :

- **`Custom elements`** : <mon-composant>
- **`Shadow DOM`** : rendu séparé du DOM
- **`HTML Templates`** : éléments réutilisables
  `<template>`
  `<slot>`

Web components

Laborieux et bas niveau, on utilise plutôt :

- Des bibliothèques comme https://lit.dev/ ;
- Des composants sur étagère https://www.webcomponents.org/ ;
- La solution imposée par les *frameworks*.

Avec lit-html

bibliothèque de https://lit.dev utilisable seule

Une méthode **sûre** (pas de XSS) pour transformer les chaînes en objets DOM :

- S'appuie sur les Tagged Templates de JS
- Crée des <template> et <slot>
- Ne rafraîchit **que** ce qui est nécessaire

```
const aliceLit = (id, source) =>
  html`
    <figure>
      <img class="alice blue" id=${id} src=${source} />
      <figcaption>Silhouette of ...</figcaption>
    </figure>
  `;

// render = calcule le diff et màj le DOM
render(aliceLit(42, "alice.svg"), root);
```

HTML - CSS - DOM

Plan

1. Rappels : composition d’une page Web
2. Le Document Object Model
3. Abstractions au dessus du DOM
4. Événements
5. Chargement et exécution du JavaScript
6. Bilan et TP

“Les événements DOM sont déclenchés pour notifier au code des « changements intéressants » qui peuvent affecter l’exécution du code. Ces changements peuvent résulter d’interactions avec l’utilisateur, […], de changements dans l’état de l’environnement […], et d’autres causes.”

MDN - Event reference

Propagation des événements

Bubbling : les événements remontent le long de l'arbre DOM des feuilles (plus précis, les plus imbriqués) à la racine.

Capture : les parents peuvent capturer les événements et ne pas les transmettre aux enfants. (*inverse du bubbling*)

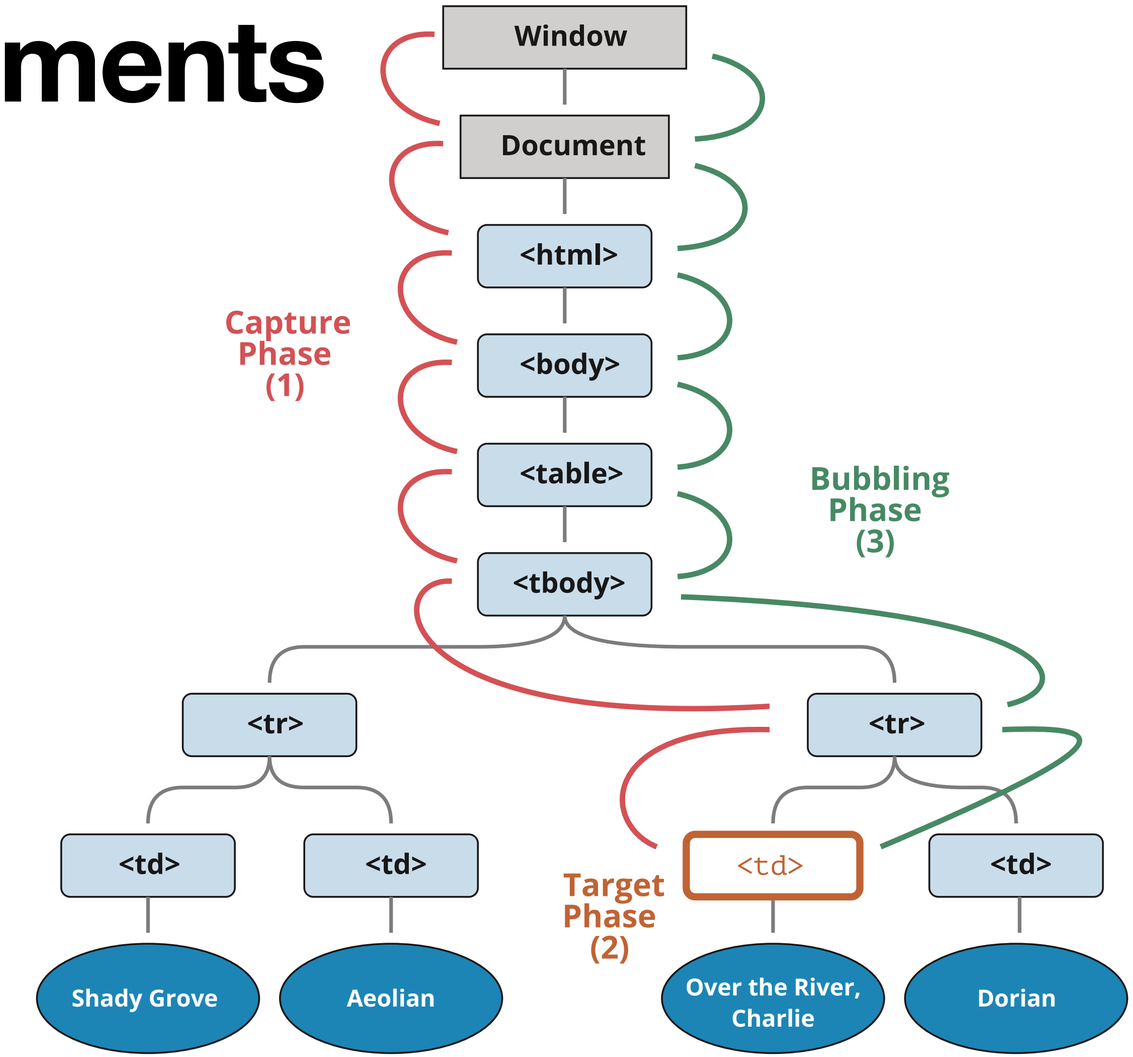
Voir MDN et javascript.info.

Propagation des événements

Bubbling est la phase “normale” de gestion des événements

`Capture` est très rarement utilisé

- Pour surveiller les rares événements qui ne *bubble* pas (`focus`, `load`…)
- Bloquer la propagation aux enfants



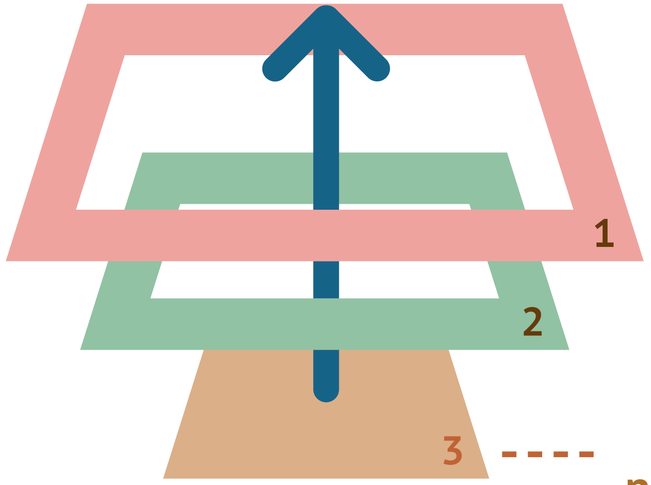
https://javascript.info/bubbling-and-capturing

Exemple de bubbling

https://javascript.info/bubbling-and-capturing

```
<style>
  body * {
    margin: 10px;
    border: 1px solid blue;
  }
</style>

<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```



Gérer les événements

> **“De nombreux éléments DOM peuvent être paramétrés afin d’accepter (« d’écouter ») ces évènements et d’exécuter du code en réaction pour les traiter (« gérer »).”**
>
> MDN - Event

- Un objet DOM (interface **EventTarget**) peut *écouter* (listen) certains événements.
- On attache une fonction appelée **handler** ou **listener** à cet événement.
- Le **handler** est appelé à chaque occurrence de l’événement.

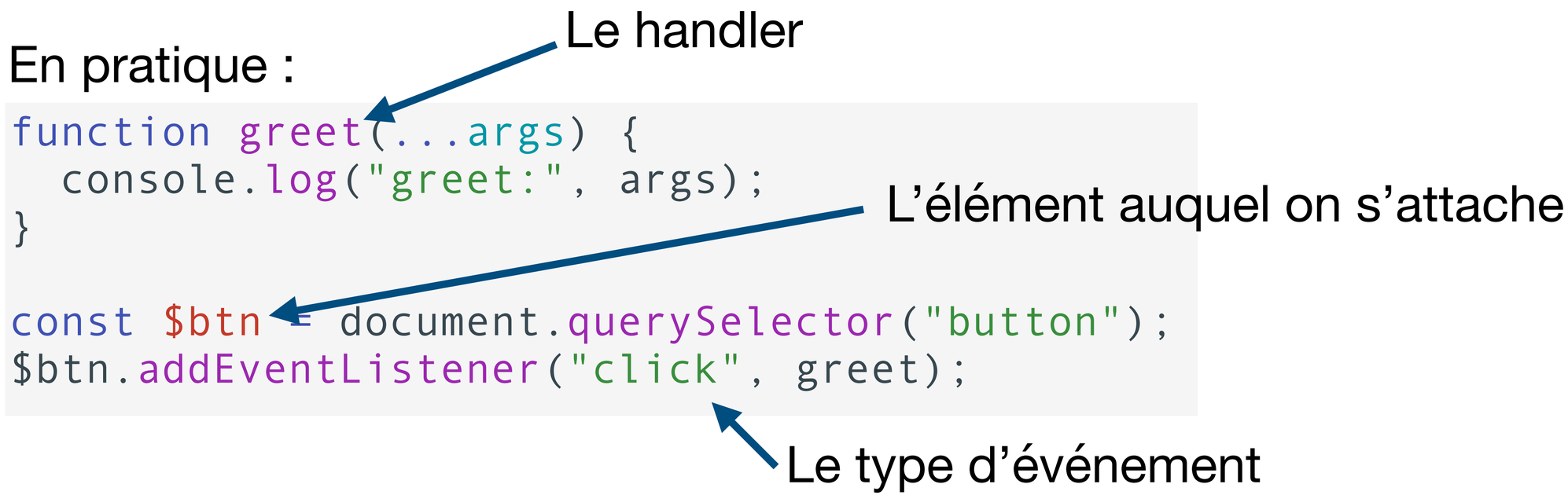
MDN - Event handling (overview)

Attacher un handler

Attacher ou détacher un handler :

```
EventTarget.addEventListener(type, handler, opts).
EventTarget.removeEventListener(type, handler, opts).
```

En pratique :




```
function greet(...args) {
  console.log("greet:", args);
}

const $btn = document.querySelector("button");
$btn.addEventListener("click", greet);
```

Attacher un handler

Méthode alternative

Attribut spécial **onevent** comme **EventTarget.onclick** :

- onevent est un attribut
- limitée à un seul handler
- plus simple à utiliser

```
function greet(...args) {
  console.log("greet:", args);
}

const $btn = document.querySelector("button");
// ajout du handler
$btn.onclick = greet;
// suprression
$btn.onclick = null;
```

On évite

Méthode inline HTML

```
<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```

☣ À bannir (MDN) ☣

☠️ le handler est une fonction ☠️

`element.onclick = handlerFunct()`

- n’est exécuté qu’**une seule fois**
  - au moment de l’assignation
  - c’est son résultat qui sera assigné
- ceci n’a pas de sens (sauf curryfication)
- provoquera (généralement) une erreur

`element.onclick = handlerFunct`

- L’appel handlerFunct() sera exécuté à chaque occurrence de l’événement "click".

On aura besoin de fermetures -> on va faire de la programmation fonctionnelle.

La source

this dans les handlers

“When a function is used as an event handler, its **this** is set to the **element on which the listener is placed**”

this and event handler | MDN

```
const $btn = document.querySelector("button");

function greet(event) {
  // affiche le texte du premier boutton de la page
  // et deux attributs de l'évenement "click"
  console.log(this.innerText, event.type, event.timeStamp);
}
$btn.addEventListener("click", greet);
```

☣️ this ≠ currentTarget ≠ target ☣️

Démo

http://lifweb.pages.univ-lyon1.fr/2024/CM2-DOM/exemples/gallerie.html

```
const $output = document.querySelector("#output");
const $div = document.querySelector("#container");
const $button = document.querySelector("button");

const handle = (event) => {
  $output.textContent += `Clicked ${event.currentTarget.tagName}\n`;
};

document.body.addEventListener("click", handle);
$div.addEventListener("click", handle);
$button.addEventListener("click", handle);
```

HTML - CSS - DOM

Plan

1. Rappels : composition d’une page Web
2. Le Document Object Model
3. Abstractions au dessus du DOM
4. Événements
5. Chargement et exécution du JavaScript
6. Bilan et TP

Stratégies de chargement du JavaScript

⚠️ directement en ligne dans le HTML

⚠️ dans l'élément <script>

👌 attribut src de l'élément <script> (MDN)

options `async` et `defer`

option `type="module"`

Options `async` et `defer`

👌 `<script defer src="...">` 👌

- Exécuté après l'analyse de toute la page
- Juste avant l'événement `DOMContentLoaded`.
- Si plusieurs defer, ils sont exécutés dans l'ordre

⚠️ `<script src="...">` ⚠️

Le script est exécuté immédiatement

Seul le contenu avant l'élément est disponible

⚠️ `<script async src="...">` ⚠️

Exécuté quand le navigateur le voudra bien…

Aucun de contrôle sur l'ordre d'exécution

Option `type="module"` **ou** `type="text/javascript"`

Chargement du JS en module (javascript.info, MDN) :

- Implicitement defer ;
- Implicitement en mode strict ;
- Exécuté au plus une fois (e.g., singleton) ;
- Espace de nom propre (pas de pollution) ;
- Autorise les top level await ;

👎 ne fonctionne pas sur `file://`

- `python3 -m http.server` ou Live Server.

HTML - CSS - DOM

Plan

1. Rappels : composition d’une page Web
2. Le Document Object Model
3. Abstractions au dessus du DOM
4. Événements
5. Chargement et exécution du JavaScript
6. Bilan et TP

Outils de développement

Devtools

- Firefox
- Chrome

https://httpie.io/ et https://curl.se/

Analyseur de protocole https://www.wireshark.org/

Syntaxe de base JS

- fichier de démonstration.
- MDN - JavaScript reference
- MDN - JavaScript
- MDN - Guide
- Wikipedia
- javascript.info - JavaScript fundamentals

Conseils généraux

Travaillez systématiquement en **navigation privée**

Utilisez des sources de référence

- https://developer.mozilla.org/
- https://javascript.info/

Validez vos documents HTML5/CSS3

- https://validator.w3.org/
- https://jigsaw.w3.org/css-validator/