

1.HTML

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Titre de la page</title>
</head>
<body>
    <header>
        <h1>Bienvenue sur mon site web</h1>
    </header>
    <nav>
        <ul>
            <li><a href="#section1">Section 1</a></li>
        </ul>
    </nav>
    <main>
        <section id="section1">
            <h2>Section 1</h2>
            <p>Contenu de la section 1.</p>
        </section>
    </main>
    <footer>
        <p>&copy; 2024 Mon site web</p>
    </footer>
</body>
</html>
```

Hyperlien et image, cependant si on se passe de l'hyperlien alors seul l'image sera affiché :

```
<a href="http://validator.w3.org/check?uri=referer">
    
</a>
```

Dans HTML:

- Les guillemets peuvent être simples, double ou absent
- Balises auto-fermant <> ou </>
- La casse n'est pas importante
- Les balises html, head et body sont optionnelles
- Pas nécessaire de spécifier une valeur pour un attribut booléen si présent alors c'est True

Attribut bool HTML :

```
<input type="checkbox" checked>
<button disabled> Click me! </button>
<input type="text" required>
```

Allowfullscreen, async, autofocus, autoplay, checked, controls, default, defer, disable, inert, ismap, multiple, muted, nomodule, open, readonly, required, selected

Il existe une centaine de tag : div, figure, figcaption, header, h1, h2, h3, h4, h5, h6, main, nav, section, aside, article, footer, canvas, map, keygen, progress, table, title, link, meta, style

Les éléments inline s'affichent dans le flux sans rupture à contrario des éléments bloc. On peut changer la propriété visuelle des blocs en mettant : display:inline et display:block

Exemple inline : audio, img, a, code, strong, script, button, iframe, embed, span, video

Attribut commun : class, id, style, name qui est réservé à :

button, form, fieldset, input, map, meta, object, output, param, select, textarea

Changements de sens :

- Balises structurantes
 - header: un ensemble pouvant inclure un titre (h1-h6), une introduction, des logos, etc.
 - footer: infos sur la section à laquelle il appartient (ex: auteur)
 - nav: liens vers d'autres pages ou d'autres parties de la page
 - aside: contenu non essentiel à une entité (ex: publicité)
 - article: portion de page qui se suffit sémantiquement
 - section: regroupement thématique de contenu contenant habituellement un titre

Permet d'avoir un temps universel

```
La réunion commence à <time datetime="2024-06-13T14:30">14h30</time>
```

La différence entre et <figure> : Simplicité vs. Richesse sémantique

- <main> is for content unique to this page. Use <main> only once per page, and put it directly inside <body>. Ideally this shouldn't be nested within other elements.

En résumé, utilise <article> pour des contenus autonomes et indépendants et <section> pour structurer et organiser le contenu thématique au sein d'un document.

L'élément <canvas> est un outil pour dessiner des graphiques et des visualisations dynamiques sur une page web. Cela permet de créer des animations, des jeux, des graphiques de données, souvent jumelé à du js

Tableaux

Characteristics with positive and negative sides		
Negative Characteristic Positive		
Sad	Mood	Happy
Failing	Grade	Passing

Note : Un tableau peut être inclus dans figure

<td headers="n r1">Sad</td> : Cellule de données avec les en-têtes "n" et "r1"

tr : ligne

th : titre dans ligne

td : cellule

La balise <label> est utilisée en HTML pour associer un libellé à un élément de formulaire

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

L'attribut for de la balise <label> est lié à l'attribut id de l'élément <input>

On peut avoir des inputs dans le tableau :

```
<th><label for=e1>Sellers Name</label>
<td><input id=e1 type="text" name="who" required form=f>
```

Formulaire :

GET : Les données sont envoyées en tant que paramètres de l'URL donc visible et limité en taille. Il est utilisé principalement pour les requêtes non-destructives (lecture de données).

POST : Les données sont envoyées dans le corps de la requête HTTP, donc pas visible + volumineux. Utilisé pour les actions qui modifient l'état du serveur

L'attribut enctype spécifie comment les données du formulaire doivent être encodées avant d'être envoyées au serveur. Il est utilisé uniquement avec la méthode POST

```
<form method="post"
      enctype="application/x-www-form-urlencoded"
      action="https://pizza.example.com/order.cgi">
```

1. Soumission automatique : Si aucun bouton de soumission (<input type="submit">, <button type="submit">, etc.) n'est présent dans le formulaire, vous pouvez toujours soumettre le formulaire à l'aide de JavaScript en appelant la méthode <submit()> sur l'élément <form> lui-même.

Exemple :

```
html
<form method="post" action="traitement.php" id="monformulaire">
    <input type="text" name="nom">
</form>

<script>
    document.getElementById("monformulaire").submit();
</script>
```

D'où l'importance de : <input type="submit" value="Envoyer">

Type input :

```

<input type="..." name="nom">
Text, password, checkbox, radio, Range min-max, Color, Checkbox, Radio, Submit,
Button, File, Number, Date, datetime-local, month, weekEmail, Url, Hidden, search
(add autofocus)

<fieldset>
<legend> Pizza Size </legend>
<p><label> <input type="radio" name="size" value="small"> Small </label></p>
<p><label> <input type="radio" name="size" value="medium"> Medium </label></p>
<p><label> <input type="radio" name="size" value="large"> Large </label></p>
</fieldset>

```

Verification cote client :

required, maxlength, minlength, pattern

```
<input type="text" pattern="^d{5,6}(?[-\s]d{4})? $" ...>
```

On peut aider l'utilisateur en ajoutant :

Placeholder et autocomplete : aide l'agent

```

<label for="frmNameA">Name</label>
<input type="text" name="name" id="frmNameA"
placeholder="Full name" required autocomplete="name">

```

Pour énumérer des suggestions :

```

<label for="frmFavChocolate">Favorite Type of Chocolate</label>
<input type="text" name="fav-choc" id="frmFavChocolate" list="chocType">
<datalist id="chocType">
<option value="white">
<option value="milk">
<option value="dark">
</datalist>

```

crédit

Input est mieux dans un form cependant il peut être utilisé en dehors

```

<form id="maforme">
  <input type="submit" />
</form>
<input type="text" name="mytext" form="maforme"/>

<form oninput="x.value=a.valueAsNumber + b.valueAsNumber">
  0 <input type="range" id="a" name="a" value="50" /> 100
  +
  <input type="number" id="b" name="b" value="50"/>
  = <output name="x" for="a b">input please ! </output>
  <br />
  <input type="reset" />
</form>

```

ids impliqués
valeur par défaut

La balise `` en HTML est un élément en ligne utilisé pour regrouper du contenu en ligne dans un document sans affecter sémantiquement le texte qu'elle contient. Voici quelques points clés sur son utilisation :

Par exemple un texte en rouge dans un paragraphe

Microdata en html :

- **itemscope** (signale un objet)
 - **itemtype** (à côté de itemscope) pour indiquer la nature de l'objet
 - **itemprop** (définit une propriété d'un objet)
- ```

<div itemscope itemtype="link">
<h1 itemprop="name">Avatar</h1>
Science fictionici on peut mettre un lien vers le
trailer et ajouter itemprop trailer

```

Un URI est une chaîne de caractères utilisée pour identifier une ressource abstraite ou concrète de manière unique.

Une URL est un type spécifique d'URI qui indique l'emplacement d'une ressource sur Internet et la manière d'y accéder.



## CSS

```

<head>
<link href="path/to/file.css" rel="stylesheet" type="text/css">
</head>
*{
margin : 0;
}

```

```

}
.classname ; #idname

/* Cible tous les éléments <input> qui ont un attribut type */
input[type] {
border: 1px solid #ccc;
padding: 5px;
}

[type="checkbox"] {
margin-right: 10px;
}

[class~=primary] {
background-color: blue;
color: white;
padding: 10px;
}
```

^ Ce sélecteur cible un élément HTML qui possède un attribut `att` dont la valeur est une liste de mots séparés par des espaces. L'un de ces mots doit correspondre exactement à "val".

```

<div class="btn primary large">Cliquez ici</div>

[lang=en] {
font-style: italic;
}
```

^ Cela cible tous les éléments qui ont l'attribut `lang` avec une valeur qui est "en" ou commence par "en-" (par exemple, "en", "en-US"). Cela est souvent utilisé pour les attributs de langue.

```

[href^="https://"] {
color: blue;
}
```

Cela cible tous les liens (`<a>`) avec l'attribut `href` dont l'URL commence par "https://", et leur applique une couleur de texte bleue.

```

[src$=".png"] {
border: 1px solid black;
}
```

Cela cible tous les éléments (par exemple, `<img>`) dont l'attribut `src` se termine par ".png", et leur applique une bordure noire de 1 pixel.

```

[title*="important"] {
background-color: yellow;
}
```

Cela cible tous les éléments qui ont un attribut `title` contenant le mot "important" dans son contenu, et leur applique une couleur de fond jaune.

Note a[att=va]

Combinaison en CSS :

**h1, h2, h3** : cible tous ces éléments

**Pintro** : Cible tous les paragraphes `<p>` avec la classe `intro`

**Div p** : Cible tous les paragraphes `<p>` qui sont descendants directs d'un élément `<div>`

**ul > li** : Cible tous les éléments `<li>` qui sont des enfants directs d'une liste non ordonnée `<ul>`

**h1 + p**: Sélecteur d'élément suivant immédiat

=> Les pseudo-classes et pseudo-éléments permettent de cibler des états spécifiques d'éléments ou des parties spécifiques d'éléments.

**a : hover**, **a** : link : lien pas encore visité donc `a:visited`, **a : active** avant que le navigateur commence à charger la nouvelle page

**P::first-line** : Cible la première ligne de tous les paragraphes `<p>`

**::before ::after ::first-line ::first-letter**

**P~span** : cible tous les éléments `<span>` qui suivent immédiatement un élément `<p>` dans le même conteneur parent.

**col || td** : permet de sélectionner tous les éléments `<td>` dans des colonnes (`<col>`) spécifiques.

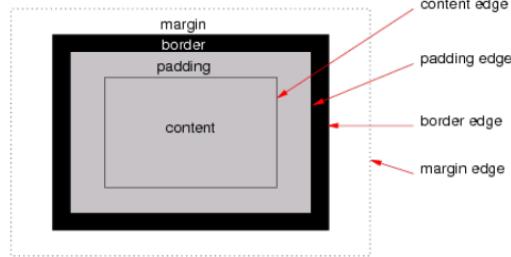
**Input:focus** : pendant que l'utilisateur tape, on a `input:enabled or disable`

**:checked**

**:root** => cible l'élément racine (`<html>` en HTML).

**:nth-child(an+b)**: sélectionner des éléments basés sur leur position dans un parent. Exemple :  $2n+1 == 1,3,5$

:first-child, :last-child, :first-of-type, :only-child, :empty => sélection en fonction de la position  
 :not(X) : sélectionner tous les éléments qui ne correspondent pas au sélecteur X  
 font-family: "Times New Roman"



## Bootstrap

| CSS files                                        | Layout           | Content      | Components   | Utilities           |
|--------------------------------------------------|------------------|--------------|--------------|---------------------|
| bootstrap.css<br>bootstrap.min.css               | Included         | Included     | Included     | Included            |
| bootstrap-grid.css<br>bootstrap-grid.min.css     | Only grid system | Not included | Not included | Only flex utilities |
| bootstrap-reboot.css<br>bootstrap-reboot.min.css | Not included     | Only Reboot  | Not included | Not included        |

|                   |     |
|-------------------|-----|
| Small             | sm  |
| Medium            | md  |
| Large             | lg  |
| Extra large       | xl  |
| Extra extra large | xxl |

```
<p class="bg-danger bg-opacity-75">Opacité à 75%</p>
```

Bg = background et danger == rouge

```
<p class="border border-3 rounded-5">L
<p class="border border-top-0 border-opacity-50">
<div class="container">
 <div class="row">
 <div class="col-md-6">Colonne 1</div>
 <div class="col-md-6">Colonne 2</div>
 </div>
</div>
```

.row : ligner les colonnes Bootstrap à l'intérieur d'un conteneur

.col : définissent les colonnes dans le système de grille Bootstrap

```
div class="alert alert-primary alert-dismissible">
 C'est un lien
```

Alert-link souligner

warning=Jaune; success = vert

Primary=bleue,Danger == Rouge

Py-2 = padding

```
<button class="btn btn-outline-primary btn-sm">Primary</button>

<div id="diaporama" class="carousel slide" data-bs-ride="carousel">
 <div class="carousel-inner">
 <div class="carousel-item active">

 </div>
 <div class="carousel-item">

 </div>
 <div class="carousel-item">

 </div>
 </div>
 <button class="carousel-control-prev" type="button" data-bs-target="#diaporama" data-bs-slide="prev">

 </button>
 <button class="carousel-control-next" type="button" data-bs-target="#diaporama" data-bs-slide="next">

 </button>
```

## Composant nav

```
<nav class="nav">
 Lien 1
```

## Grille responsive

```
<!--
 Le col-12 pourrait être facultatif
-->
<div class="col-12 col-md-6 col-lg-3">
```

```
<div class="w-25 float-start">

</div>
```

## DOM

il ne devrait pas y avoir de script javascript ailleurs que dans l'élément

```
<body onload="init();">
 ...
</body>
```

Le script sera exécuté après le chargement de la page

Autre facon :

Dans le head :

```
<script type="text/javascript" src=<monscript>.js></script>
```

Dans js :

```
window.addEventListener("load", init, false);
```

```
var ret = document.forms[0];
alert("Document First Form : " + ret);
var ret = document.forms[0].elements[1];
alert("Second element : " + ret);
```

Parmi les interfaces du DOM, quelques-unes vont particulièrement nous intéresser :

- L'interface **Window** qu'on a déjà étudié et qui est liée au DOM ;
- L'interface **Event** qui représente tout événement qui a lieu dans le DOM (nous allons définir précisément ce qu'est un événement dans la suite de cette partie) ;
- L'interface **EventTarget** qui est une interface que vont implémenter les objets qui peuvent recevoir des événements ;
- L'interface **Node** qui est l'interface de base pour une grande partie des objets du DOM ;
- L'interface **Document** qui représente le document actuel et qui va être l'interface la plus utilisée ;
- L'interface **Element** qui est l'interface de base pour tous les objets d'un document ;

L'interface **EventTarget** est l'interface parent de **Node** et donc **Node** hérite (des propriétés et méthodes) de l'interface **EventTarget** ;

- L'interface **Node** est le parent des interfaces **Document** et **Element** qui héritent donc de **Node** (et donc par extension également de **EventTarget**). De plus, **Document** et **Element** implémentent les mixin **ParentNode** et **ChildNode** ;
- L'interface **Element** implémente également le mixin **NonDocumentTypeChildNode** ;
- L'interface **Document** implémente également le mixin **NonElementParentNode** ;
- L'interface **HTMLElement** hérite de l'interface **Element**

## Accéder à un élément à partir de son sélecteur CSS associé

```
// Sélectionne tous les paragraphes du document
let documentParas = document.querySelectorAll('p');
```

```
/* Sélectionne le premier paragraphe du document et change son texte avec la propriété textContent que nous étudierons plus tard dans cette partie */
document.querySelector('p').textContent = '1er paragraphe du document';
```

## Accéder à un élément en fonction de la valeur de son attribut id

```
// Sélectionne l'élément avec un id = 'p1' et modifie la couleur du texte
document.getElementById('p1').style.color = 'blue';
```

## Accéder à un élément en fonction de la valeur de son attribut class

```
//Sélectionne les éléments avec une class = 'bleu'
let bleu = document.getElementsByClassName('bleu');

// "bleu" est un objet de HTMLCollection qu'on va manipuler comme un tableau
for(valeur of bleu){
 valeur.style.color = 'blue';
}
```

En effet, ça retourne une liste

Accéder à un élément en fonction de son identité

```
//Sélectionne tous les éléments p du document
let paras = document.getElementsByTagName('p');
```

`getElementsByName()` qui renvoie un objet `NodeList` contenant la liste des éléments portant un attribut `name` avec la valeur spécifiée en argument sous forme d'objet

=> Quiz : Mettre le script à la fin du document HTML, juste avant la fermeture de la balise `</body>`, est une pratique courante pour plusieurs raisons : Assurer que le DOM est complètement chargé, Améliorer les performances de chargement de la page et Éviter les erreurs de script

```
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) inside "main" is: ' + y[0].innerHTML;
```

La propriété `innerHTML` d'un élément DOM est utilisée pour obtenir ou définir le contenu HTML à l'intérieur de cet élément. Dans ce cas, nous utilisons `innerHTML` pour définir le contenu de l'élément `<p id="demo">`.

```


<script>
document.getElementById("image").src = "landscape.jpg";
</script>
```

La propriété `innerHTML` de l'interface `Element` permet de récupérer ou de redéfinir la syntaxe HTML interne à un élément ;

- La propriété `outerHTML` de l'interface `Element` permet de récupérer ou de redéfinir l'ensemble de la syntaxe HTML interne d'un élément et de l'élément en soi ;

- La propriété `textContent` de l'interface `Node` représente le contenu textuel d'un nœud et de ses descendants. On utilisera cette propriété à partir d'un objet `Element` ;
- La propriété `innerText` de l'interface `Node` représente le contenu textuel visible sur le document final d'un nœud et de ses descendants. On utilisera cette propriété à partir d'un objet `Element`

```
//Accède au contenu HTML interne du div et le modifie
document.querySelector('div').innerHTML +=
 'Elément n°1Elément n°2';

//Accède au HTML du 1er paragraphe du document et le modifie
document.querySelector('p').outerHTML = '<h2>Je suis un titre h2</h2>';

/*Accède au contenu textuel de l'élément avec un id='texte' et le modifie.
Les balises HTML vont ici être considérées comme du texte/
document.getElementById('texte').textContent = 'Texte modifié';

//Accède au texte visible de l'élément avec l'id = 'p2'
let texteVisible = document.getElementById('p2').innerText;
//Accède au texte (visible ou non) de l'élément avec l'id = 'p2'
let texteEntier = document.getElementById('p2').textContent;

//Affiche les résultats du dessus dans l'élément avec l'id = 'p3'
document.getElementById('p3').innerHTML =
 'Texte visible : ' + texteVisible + '
Texte complet : ' + texteEntier;

let p1 = document.getElementById('p1');
let p2 = document.getElementById('p2');

p2.parentNode.style.backgroundColor = 'RGB(240,160,000,0.5)'; //Orange

//On accède à tous les noeuds enfants de p1. childNodes renvoie une NodeList
let enfantsP1 = p1.childNodes;

/*On peut ensuite utiliser une boucle forEach() pour tous les manipuler
*un indice comme pour les tableaux pour manipuler un nœud enfant en particulier (le premier enfant à l'indice 0, le deuxième à l'indice 1, etc)
enfantsP1[1].style.fontWeight = 'bold';

/*On accède aux noeuds enfants éléments seulement de p1.
children renvoie une HTMLCollection/
let enfantsElp1 = p1.children;
```

```
//On accède au premier noeud enfant de body
let bodyFirstChild = document.body.firstChild;

//On accède au dernier noeud enfant de body
let bodyLastChild = document.body.lastChild;

//On accède au premier noeud enfant élément de body
let bodyFirstElementChild = document.body.firstElementChild;

//On accède au dernier noeud enfant élément de body
let bodyLastElementChild = document.body.lastElementChild;
```

Créer un noeud Principe: dès qu'un noeud est ajouté/modifié/supprimé dans la DOM, le navigateur affiche à nouveau la page

```
var n = document.createElement('h4');
n.appendChild(document.createTextNode('mon titre'));
```

`document.createElement`: Cette méthode est utilisée pour créer un nouvel élément HTML spécifié par le nom de la balise passée en argument.

`document.createTextNode`: Cette méthode crée un nouveau nœud de texte contenant la chaîne de caractères spécifiée ('mon titre' dans ce cas).

`n.appendChild`: Cette méthode est utilisée pour ajouter un nœud enfant à l'élément référencé par `n`. Le nœud enfant ajouté ici est le nœud de texte créé précédemment.

Ensuite on insère :

```
document.getElementsByTagName('body').item(0).insertBefore(n,document.getElementsByTagName('p').item(0));
```

`document.getElementsByTagName('body')`: Cette méthode retourne une collection de tous les éléments `<body>` du document. Bien qu'il n'y ait généralement qu'un seul élément `<body>` dans un document HTML bien formé, cette méthode retourne toujours une collection.

`document.getElementsByTagName('p')`: Cette méthode retourne une collection de tous les éléments `<p>` du document

`insertBefore(newNode, referenceNode)`: Cette méthode insère `newNode` dans le DOM avant `referenceNode`.

`n`: Est l'élément à insérer, dans ce cas, c'est notre nouvel élément `<h4>`

```
element.addEventListener(event, function, useCapture);
```

`event`: Une chaîne de caractères représentant le type d'événement (par exemple, 'click', 'mouseover', 'keydown').

`useCapture (optionnel)`: Un booléen ou un objet qui indique si l'événement doit être capturé pendant la phase de capture ou de propagation. Par défaut, c'est false, ce qui signifie que l'événement est écouté pendant la phase de propagation.

```
// Sélection de l'élément bouton
var button = document.getElementById('myButton');

// Fonction de rappel qui sera exécutée lorsque l'événement se déclenche
function handleClick() {
 alert('Bouton cliqué !');
}

// Attachement de l'événement 'click' au bouton
button.addEventListener('click', handleClick);
```

Créer de nouveaux nœuds et les ajouter dans l'arborescence du DOM

```
let newP = document.createElement('p');
newP.textContent = 'Paragraphe créé et inséré grâce au JavaScript';
```

```
let b = document.body;
let newP = document.createElement('p');
let newTexte = document.createTextNode('Texte écrit en JavaScript');

newP.textContent = 'Paragraphe créé et inséré grâce au JavaScript';

//Ajoute le paragraphe créé comme premier enfant de l'élément body
b.prepend(newP);

//Ajoute le texte créé comme dernier enfant de l'élément body
b.append(newTexte);
```

La méthode `append()` permet également d'ajouter directement une chaîne de caractères tandis que `appendChild()` n'accepte que des objets de type `Node` ;

- La méthode `append()` peut ajouter plusieurs nœuds et chaînes de caractères au contraire de `appendChild()` qui ne peut ajouter qu'un nœud à la fois ;

- La méthode `append()` n'a pas de valeur de retour, tandis

que appendChild() retourne l'objet ajouté.

```
let b = document.body;
let p1 = document.getElementById('p1');
let newP = document.createElement('p');

newP.textContent = 'Paragraphe créé et inséré grâce au JavaScript';

b.insertBefore(newP,p1);
```

On peut encore utiliser la méthode `insertBefore()` de l'interface Node qui permet pour sa part d'insérer un nœud en tant qu'enfant d'un autre nœud juste avant un certain nœud enfant donné de ce parent.

Pour Déplacer eux elements

```
let b = document.body;
let p1 = document.getElementById('p1');
let p4 = b.lastElementChild; //On accède au dernier paragraphe

//On déplace p1 juste avant p4 dans le DOM
p.insertBefore(p1, p4);
```

```
const btn = document.querySelector('button');
const videoBox = document.querySelector('div');

function displayVideo() {
 if (videoBox.getAttribute('class') === 'hidden') {
 videoBox.setAttribute('class', 'showing');
 }
}

btn.addEventListener('click', displayVideo);
videoBox.addEventListener('click',
 () => videoBox.setAttribute('class', 'hidden'));
const video = document.querySelector('video');
video.addEventListener('click', () => video.play());

L'attribut onclick pour l'événement « clic sur un élément »;
L'attribut onmouseover pour l'événement « passage de la souris sur un élément »;
;
L'attribut onmouseout pour l'événement « sortie de la souris d'un élément »;
```

Pour stopper la propagation d'un événement, on va pouvoir utiliser la méthode `stopPropagation()` de l'interface Event.

```
video.onclick = function(e) {
 e.stopPropagation();
 video.play();
};
```

Supposons qu'on a un élément `<div>` contenant un `<button>`, et qu'on clique sur le bouton. La propagation de l'événement de clic se fait comme suit :

- **Phase de capture** : document -> body -> div -> button.
- **Phase de bouillonnement** : button -> div -> body -> document.

```
document.addEventListener('DOMContentLoaded', function() {
 var cell, row;
 row = document.getElementsByTagName('tr')[0];

 // Vérifier si les cellules existent déjà dans le stockage local
 if (!localStorage.getItem('factorials')) {
 var factorials = [];
 var res = 1;
 for (var i = 1; i < 6; i++) {
 res *= i;
 factorials.push(res);
 }
 localStorage.setItem('factorials', JSON.stringify(factorials));
 }

 // Récupérer les cellules stockées et les ajouter à la table
 var storedFactorials = JSON.parse(localStorage.getItem('factorials'));
 storedFactorials.forEach(function(factorial) {
 cell = document.createElement('td');
 cell.textContent = factorial;
 row.appendChild(cell);
 });
});
```

## PHP

Lorsqu'on intègre du PHP dans du code HTML, on va pouvoir placer cette balise et du code PHP A n'importe quel endroit dans notre fichier. On va même pouvoir placer la balise PHP en dehors de notre élément `html`. De plus, on va pouvoir déclarer plusieurs balises PHP à différents endroits dans un fichier

Toute variable en PHP doit commencer par le signe \$ qui sera suivi du nom de la variable ; De plus, notez que le nom des variables est sensible à la casse en PHP.

- Le type « chaîne de caractères » ou `String` en anglais ;
- Le type « nombre entier » ou `Integer` en anglais ;
- Le type « nombre décimal » ou `Float` en anglais ;
- Le type « booléen » ou `Boolean` en anglais ;
- Le type « tableau » ou `Array` en anglais ;
- Le type « objet » ou `Object` en anglais ;
- Le type « NULL » qui se dit également `NULL` en anglais ;
- Le type « ressource » ou `Resource` en anglais ;

```
<body>
 <h1>Titre principal</h1>
 <?php
 $prez = "Je m'appelle Pierre";
 $age = 28; // Stocke le nombre 28
 $age2 = "28"; // Stocke la chaîne de caractères "28"
 $distance = 2.84;
 $vrai = true;
 $faux = false;

 echo "La variable \$vrai contient une valeur de type ";
 echo gettype($vrai);

 echo "
La variable \$faux contient une valeur de type ";
 echo gettype($faux);
 ?>
```

Par exemple, l'opérateur + va nous permettre d'additionner les valeurs de deux variables

Les opérateurs logiques		
Exemple	Nom	Résultat
\$a and \$b	And (Et)	true si \$a ET \$b valent true.
\$a or \$b	Or (Ou)	true si \$a OU \$b valent true.
\$a xor \$b	XOR	true si \$a OU \$b est true, mais pas les deux en même temps.
!\$a	Not (Non)	true si \$a n'est pas true.
\$a && \$b	And (Et)	true si \$a ET \$b sont true.
\$a    \$b	Or (Ou)	true si \$a OU \$b est true.

Comparaison large avec ==												
true	false	1	0	-1	"1"	"0"	"-1"	null	[]	"php"	" "	
true	false	true	false	true	true	false	true	false	false	true	false	
false	true	false	true	false	false	false	true	false	true	false	true	
1	true	false	true	false	false	true	false	false	false	false	false	
0	false	true	false	true	false	false	true	false	false	false	"false"	
-1	true	false	false	true	false	false	true	false	false	false	false	
"1"	false	true	false	true	false	false	true	false	false	false	false	
"0"	false	true	false	true	false	false	true	false	false	false	false	
"-1"	true	false	false	true	false	false	true	false	false	false	false	
null	false	true	false	true	false	false	true	true	false	true	false	
[]	false	true	false	false	false	false	false	true	true	false	false	
"php"	true	false	true	false								
" "	false	true	false	true								

Opérateur	Définition
==	Permet de tester l'égalité sur les valeurs
==!=	Permet de tester l'égalité en termes de valeurs et de types
!=	Permet de tester la différence en valeurs
<>	Permet également de tester la différence en valeurs
!=~	Permet de tester la différence en valeurs ou en types
<~	Permet de tester si une valeur est strictement inférieure à une autre
>	Permet de tester si une valeur est strictement supérieure à une autre
<=	Permet de tester si une valeur est inférieure ou égale à une autre
>=	Permet de tester si une valeur est supérieure ou égale à une autre

echo va transformer toute valeur en chaîne de caractères avant de l'afficher. Or true = 1. Le rôle de la fonction PHP `var_dump()` est en effet d'afficher les informations structurées d'une variable ou d'une expression et notamment son type et sa valeur. C'est pour cela que le navigateur nous affiche des `bool(true)` ou `bool(false)`.

```
$x = 4; //On affecte la valeur 4 à $x
$y = 2; //On affecte la valeur 2 à $y

if($x > 1){
 echo '$x contient une valeur stric. supérieure à 1
';
}elseif($x == 1){
 echo '$x contient la valeur 1
';
}else{
 echo '$x contient une valeur stric. inférieure à 1
';
}
```

```

for($x = 0; $x <= 5; $x++){
 echo '$x contient la valeur ' . $x. '
';
}

function test($a, $b){
 echo $a. ' + ' . $b. ' = ' . ($a + $b). '
';
}

function addition(float $a, float $b){
 echo $a. ' + ' . $b. ' = ' . ($a + $b). '
';
}

test(3, 4);

$prenoms = array('Mathilde', 'Pierre', 'Amandine', 'Florian');
$ages = [27, 29, 21, 29];

$taille = count($prenoms);

//On peut soit parcourir le tableau
for($i = 0; $i < $taille; $i++) {
 echo $prenoms[$i]. ', ';
}

```

#### TABLEAU ASSOCIATIF

```

$ages = ['Mathilde' => 27, 'Pierre' => 29, 'Amandine' => 21];

/*Identique à
*$ages = array('Mathilde' => 27, 'Pierre' => 29, 'Amandine' => 21);
 */

$mails['Mathilde'] = 'math@gmail.com';
$mails['Pierre'] = 'pierre.giraud@edhec.com';
$mails['Amandine'] = 'amandine@lp.fr';

echo 'Pierre a ' . $ages['Pierre']. ' ans
';

```

```

foreach($ages as $clef => $valeur){
 echo $clef. ' a ' . $valeur. ' ans
';
}

```

throw new LogicException('Step must be positive');

```

<style>
 div, p, fieldset, input {
 background-color: #e6e6fa;
 }
 div p, a {
 background-color: #2e6b2e;
 }
 body div p {
 background-color: #007bff;
 }
 p {
 background-color: #dc3545;
 }
 .exam {
 background-color: #800080;
 }
</style>
<head>
<body>
 <div>
 <p id="1">Voici un paragraphe. blue</p>
 </div>
 <p id="2">Un autre paragraphe. pink</p>
 <p class="exam" id="3">Encore un autre paragraphe purple</p>

```

```

<?php
session_start();
$_SESSION['username'] = 'john_doe';
$_SESSION['email'] = 'john@example.com';
?>

<?php
foreach ($_GET as $k => $v)
 echo "$k == $v". '
' ;
?>

<?php
foreach ($_POST as $k => $v)
 echo "$k == $v". '
' ;
?>

```

```

<form action="radio_button.php" method="post">
 <input type="checkbox" name="check_list[]" value="value 1">
 <input type="checkbox" name="check_list[]" value="value 2">
 <input type="checkbox" name="check_list[]" value="value 3">
 <input type="checkbox" name="check_list[]" value="value 4">
 <input type="checkbox" name="check_list[]" value="value 5">
 <input type="submit" />
</form>

```

```

<?php
$br = "
";
if(!empty($_POST['check_list'])) {
 foreach($_POST['check_list'] as $check) {
 echo $check.$br; //echoes the value set in the HTML form for each checked checkbox.
 //so, if I were to check 1, 3, and 5 it would echo value 1, value 3, value
5.
 }
}
?>

```

**accueil.php (exec)**

```

<form action="session-shop.php" method="post">
 <div class="form-group">
 <label for="username">Username:</label>
 <input type="text" placeholder="Enter Username" name="username" required>
 </div>
 <label for="password">Password:</label>
 <input type="password" placeholder="Enter Password" name="password" required>
 <button type="submit">Login</button>
</form>

```

**session-shop.php (exec)**

```

<?php
session_start();
...
<?php
if($_POST['check']){
 echo "ca va counter cher. illico a href=\"cash.php\">ic
 i" ;
} else {
 echo "form action=\"session-shop.php\" method=\"post\">
 <input type=\"hidden\" name=\"id\" value=" . $id . " />
 &name=>0.33)>
 foreach ($t as $k=>$v) {
 $_SESSION[$k] = $v;
 }
 echo "label from \"$id\">$k</label>";
 echo "<input type=\"checkbox\" name=\"$product[]\" value=" . $k . " />
 >" value="3K" id="3K" />
 echo "<hr>";
 echo "<input type=\"submit\" value=\"add\" />";
 echo "<input name=\"check\" type=\"submit\" value=\"checkout\" />
 ?>
 echo "</form></fieldset>";
}
?>

```

**Login Form**

Username	<input type="text"/>
	<small>Enter Username</small>
Password	<input type="password"/>

```

session_start();

// Vérifie si 'count' n'existe pas dans la session, et l'initialise à 1 si c'est le cas
if(empty($_SESSION['count'])) {
 $_SESSION['count'] = 1;
} else {
 // Incrémente 'count' à chaque visite
 $_SESSION['count']++;
}

// Génère une clé aléatoire de 5 caractères et l'ajoute à la session
$randomKey = substr(str_shuffle("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"), 0, 5);
$_SESSION[$randomKey] = true;
}
?>
<!DOCTYPE html>
<html>
<head>
<title>Compteur de visites</title>
</head>
<body>
<p>
 Bonjour visiteur, vous avez vu cette page <?php echo $_SESSION['count']; ?> fois.
</p>
<!-- Affiche l'ID de session
echo "session_id: " . session_id(); -->


```

```

// Affiche le cookie de session (PHPSESSID)
echo "phpsessid: " . $_COOKIE['PHPSESSID'] . "

";

// Affiche toutes les variables de session
foreach ($_SESSION as $k => $v) {
 echo "$k => $v
";
}

<?php
session_start();
session_destroy();
?>

<html>
<p>
 Revenir sur session-on.php
</p>
</html>

```

```

// Affiche le cookie de session (PHPSESSID)
echo "phpsessid: " . $_COOKIE['PHPSESSID'] . "

";

// Affiche toutes les variables de session
foreach ($_SESSION as $k => $v) {
 echo "$k => $v
";
}

<?php

```

**redirect.php (exec)**

```

<?php
header("Location: https://my.bluehost.com/hosting/help/241/"); ?>

```

**account.sql**

```

-- Table structure for table `accounts`
--
CREATE TABLE `accounts` (
 `account_id` int(10) unsigned NOT NULL,
 `account_name` varchar(255) NOT NULL,
 `account_passwd` varchar(255) NOT NULL,
 `account_reg_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `account_enabled` tinyint(1) UNSIGNED NOT NULL DEFAULT '1',
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

-- Indexes for table `accounts`

ALTER TABLE `accounts` ADD PRIMARY KEY (`account\_id`), ADD UNIQUE KEY `account\_name` (`account\_name`);

-- AUTO\_INCREMENT for table `accounts`

ALTER TABLE `accounts` MODIFY `account\_id` int(10) UNSIGNED NOT NULL AUTO\_INCREMENT;

-- Table structure for table `account\_sessions`

CREATE TABLE `account\_sessions` (
 `session\_id` varchar(255) NOT NULL,
 `account\_id` int(10) UNSIGNED NOT NULL,
 `login\_time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP,
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- Indexes for table `account\_sessions`

ALTER TABLE `account\_sessions` ADD PRIMARY KEY (`session\_id`);

```

<?php
/* The PDO object */
$pdo = NULL;

/* Connection string, or "data source name" */
$dsn = 'mysql:host=' . $host . ';dbname=' . $schema;

/* Connection inside a try/catch block */
try {
 /* PDO object creation */
 $pdo = new PDO($dsn, $user, $passwd);

 /* Enable exceptions on errors */
 $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch (PDOException $e)
{
 /* If there is an error an exception is thrown */
 echo 'Database connection failed.';
 die();
}

/*
 * Logout the current user */
public function logout()
{
 global $pdo;

 if (is_null($this->id))
 return;

 /* If there is an open Session, remove it from the account_sessions table */
 if ($session_status == PHP_SESSION_ACTIVE)
 {
 $query = 'DELETE FROM '.$schema.'.account_sessions WHERE (session_id = :sid)';
 $values = array(':sid' => session_id());
 try{
 $res = $pdo->prepare($query);
 $res->execute($values);
 }
 catch (PDOException $e) {
 throw new Exception('Database query error');
 }
 }
}

read-account.php [exec]

<?php
/* Include the database connection file (remember to
require './db_inc.php';

$stmt = $pdo->prepare("SELECT * FROM `accounts`");
$stmt->execute();
$arr = $stmt->fetchAll(PDO::FETCH_ASSOC);

if (!empty($arr)) exit('No rows');

print_r($arr);
$stmt = null;
?>

/* Login with username and password */
public function login(string $name, string $passwd): bool
{
 global $pdo;
 // verifications (skipped)

 $query = 'SELECT * FROM '.$schema.'.accounts WHERE (account_name = :name) AND (account_enabled = 1)';
 $values = array(':name' => $name);

 try {
 $res = $pdo->prepare($query);
 $res->execute($values);
 }
 catch (PDOException $e) {
 throw new Exception('Database query error');
 }

 $row = $res->fetch(PDO::FETCH_ASSOC);

 if (!empty($row)) {
 if ($password_verify($passwd, $row['account_passwd'])) {
 $this->id = intval($row['account_id'], 10);
 $this->name = $name;
 $this->authenticated = TRUE;

 $this->registerLoginSession(); ← crée/modifie entrée dans account_sess
 return TRUE;
 }
 }
}

/* If we are here, it means the authentication failed: return FALSE */
return FALSE;
}

/*
 * Logout the current user */
public function logout()
{
 global $pdo;

 if (is_null($this->id))
 return;

 /* If there is an open Session, remove it from the account_sessions table */
 if ($session_status == PHP_SESSION_ACTIVE)
 {
 $query = 'DELETE FROM '.$schema.'.account_sessions WHERE (session_id = :sid)';
 $values = array(':sid' => session_id());
 try{
 $res = $pdo->prepare($query);
 $res->execute($values);
 }
 catch (PDOException $e) {
 throw new Exception('Database query error');
 }
 }
}

/* Saves the current Session ID with the account ID */
private function registerLoginSession()
{
 /* Global $pdo object */
 global $pdo;

 /* Check that a Session has been started */
 if ($session_status == PHP_SESSION_ACTIVE)
 {
 /* Use a REPLACE statement to:
 - inserts a new row with the session id, if it doesn't exist, or...
 - update the existing row having the session id, if it does exist.
 */
 $query = 'REPLACE INTO '.$schema.'.account_sessions (session_id, account_id, login_time) VALUES
($sid, :account_id, NOW())';
 $values = array(':id' => $session_id, ':account_id' => $this->id);
 try{
 $res = $pdo->prepare($query);
 $res->execute($values);
 }
 catch (PDOException $e) {
 /* If there is a PDO exception, throw a standard exception */
 throw new Exception('Database query error');
 }
 }
}

db_inc.php

```

session\_start(); → log-account

```

/* Include the database connection file (remember to change the connection parameters) */
require './db_inc.php';

/* Include the Account class file */
require './account_class.php';

/* Create a new Account object */
$account = new Account();

/* Generate logs */
require './log-account.php';
// must be provided when needed
$query = $_GET['q'] ?? 'help';
$user = $_GET['user'] ?? 'elpipe_123';
$pwd = $_GET['pwd'] ?? '987654321';
$aid = $_GET['account_id'] ?? 1;

// Log
error_log(
 sprintf("--- query: %s user: %s pwd: %s aid: %s",
 $query, $user, $pwd, $aid)
);

if ($query == 'add') {
 // 1. Insert a new account (execute twice to test the "already existing" account error)
 try{
 $newId = $account->addAccount($user, $pwd);
 }
 catch (Exception $e)
 {
 echo $e->getMessage();
 die();
}

<?php
session_start(); // démarrer ou reprendre une session

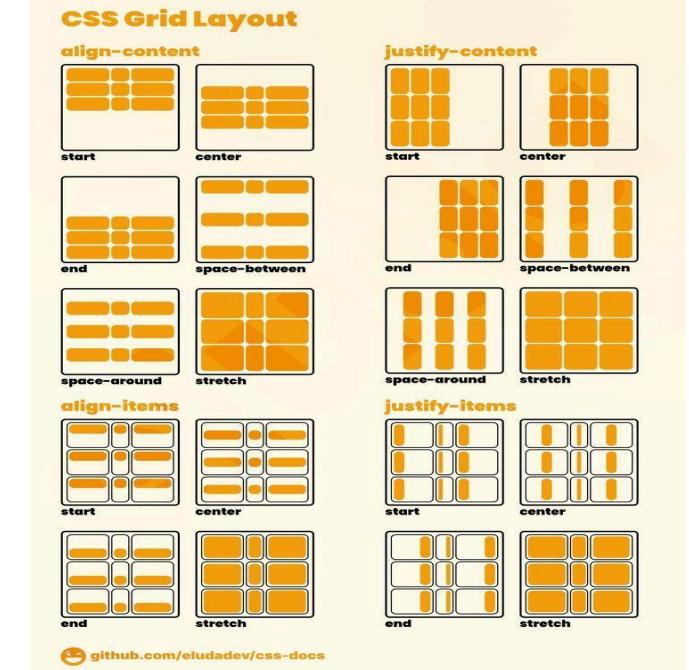
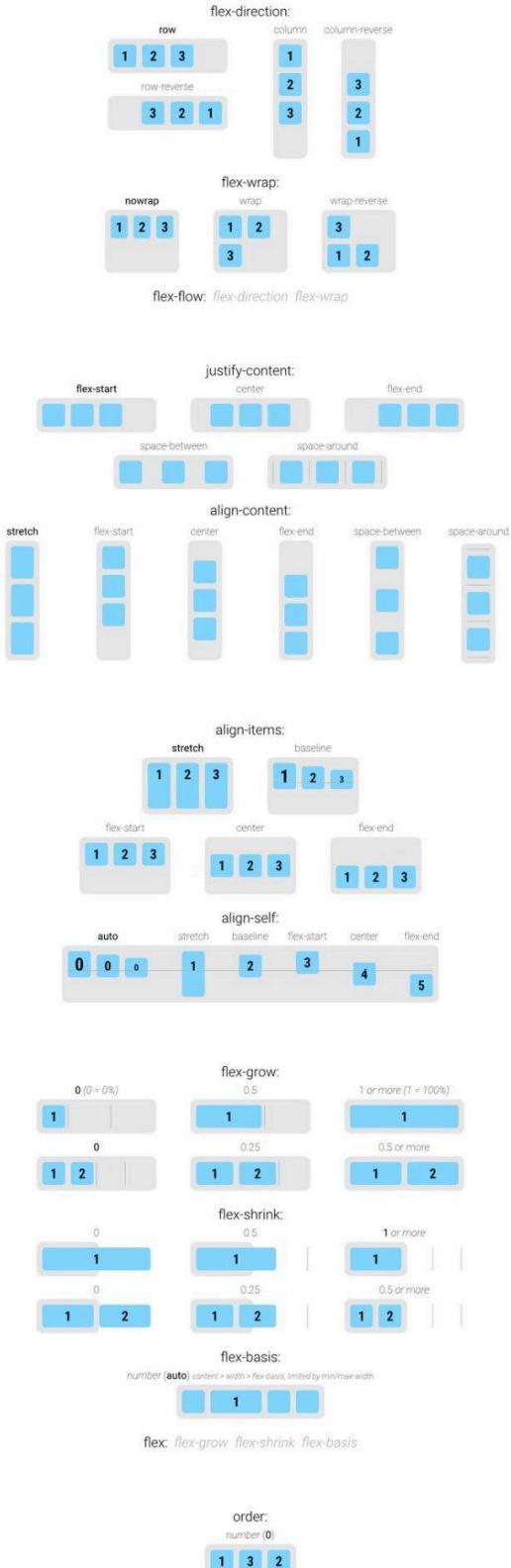
// vérifier si une variable de session spécifique existe
if (isset($_SESSION['user'])) {
 // Si la session existe, afficher un certain contenu
 <?php
 <!DOCTYPE html>
 <html>
 <head>
 <title>Page pour utilisateurs connectés</title>
 </head>
 <body>
 <h1>bienvenue, <?php echo htmlspecialchars($_SESSION['user']); ?>!</h1>
 <p>Vous êtes connecté.</p>
 </body>
 </html>
 </?php
} else {
 // Si la session n'existe pas, afficher un autre contenu
 <?php
 <!DOCTYPE html>
 <html>
 <head>
 <title>Page de connexion</title>
 </?php
}

if ($_SERVER['REQUEST_METHOD'] == "POST") {
 $email = $_POST['email'];
 $password = $_POST['pass'];
 echo $email." - ".$password."
";

 if ($email != "" && $password != "") {
 $token = bin2hex(random_bytes(length: 32));
 $requete = $bdd->query(query: "SELECT * FROM users WHERE email = '$email' AND mdp = '$password'");
 $response = $requete->fetch();
 echo "Message 1
";
 if ($response['id'] != false) {
 echo "Message 2
";
 $bdd->exec(statement: "UPDATE users SET token = '$token' WHERE email = '$email' AND mdp = '$password'");
 setcookie("token", $token, time() + 3600);
 setcookie("email", $email, time() + 3600);
 echo "Message 3
";
 header(header: "Location: client.php");
 exit();
 } else {
 $error_msg = "Informations incorrectes. Veuillez réessayer !
";
 }
 }
}

```

# FLEXBOX



Apache (officiellement Apache HTTP Server) est un logiciel de serveur web open-source. C'est l'un des serveurs web les plus populaires et les plus utilisés dans le monde.

- Quand un utilisateur entre une URL dans son navigateur, la requête est envoyée au serveur Apache.
- Apache traite cette requête et renvoie la page web demandée

JSON ("clé": valeur)

JSON est une collection de paires clé/valeur où la clé est une chaîne placée entre guillemets « " ». La clé est séparée de la valeur par un « : » (deux points)

- Number;
- String (chaîne de caractères);
- Boolean (une valeur booléenne);
- Array (un tableau);
- Object (un objet);
- null (la valeur nulle).

Pour convertir une chaîne JSON en objet JavaScript on utilise: `const usager = JSON.parse('{"nom": "Bob"}');`

Pour convertir un objet JavaScript en chaîne JSON on utilise: `const texte = JSON.stringify(usager);`

AJAX permet de faire des requêtes asynchrones au serveur sans recharger toute la page.

*Voici les étapes typiques pour créer une requête AJAX en utilisant XMLHttpRequest :*

```
// Création de l'objet XMLHttpRequest
var xhr = new XMLHttpRequest();

// Configuration de la requête
xhr.open("GET", "url_du_serveur", true);

// Définition de la fonction de callback
xhr.onreadystatechange = function() {
 if (xhr.readyState == 4 && xhr.status == 200) {
 // Traitement de la réponse
 console.log(xhr.responseText);
 }
};

// Envoi de la requête
xhr.send();
```

- AJAX est asynchrone, ce qui signifie que le code continue à s'exécuter pendant que la requête est en cours.

- Il faut gérer correctement les états de chargement et les erreurs potentielles.

- Les anciennes versions d'Internet Explorer utilisaient un objet différent (ActiveXObject) pour AJAX.

La méthode GET dans AJAX est utilisée pour récupérer des données du serveur sans modifier l'état du serveur.

autres "hooks":	<i>le cœur d'élément comme ça.</i>
event handler	event handler event type
onloadstart	loadstart
onprogress	progress
onabort	abort
onerror	error
onload	load
ontimeout	timeout
onloadend	loadend

```
// states
const unsigned short UNSENT = 0;
const unsigned short OPENED = 1;
const unsigned short HEADERS_RECEIVED = 2;
const unsigned short LOADING = 3;
const unsigned short DONE = 4;
readonly attribute unsigned short readyState;
```

The following is the event handler (and its corresponding event handler object) by the XMLHttpRequest object:

```
<!DOCTYPE html>
<html lang="fr">
<head>
 <meta charset="UTF-8">
 <title>Exemple AJAX GET</title>
 <script src="script.js"></script>
</head>
<body>
 <form action="" onsubmit="submitForm(event); return false;">
 <input type="submit" value="Récupérer le contenu">
 <input type="text" id="dyn" size="40" value="">
 </form>
</body>
</html>
```

```
function createXHR() {
 return new XMLHttpRequest();
}

function submitForm(e) {
 e.preventDefault(); // Empêche le formulaire de se soumettre normalement

 var xhr = createXHR();
 xhr.onreadystatechange = function() {
 if (xhr.readyState === 4) { // 4 signifie que la requête est terminée
 if (xhr.status === 200) {
 // Succès : on affiche la réponse
 document.getElementById('dyn').value = xhr.responseText;
 } else {
 // Erreur : on affiche le code d'erreur
 document.getElementById('dyn').value = "Erreur : " + xhr.status;
 }
 }
 };

 // Établir la connexion
 xhr.open("GET", "hello.txt", true); // true pour asynchrone
 xhr.send();
}

var donnees = JSON.parse(xhr.responseText);
// Afficher les données dans le HTML
document.getElementById("resultat").innerHTML =
 "Nom : " + donnees.nom + "
" +
 "Texte : " + donnees.texte;
// ---- ou -----
document.getElementById("resultat").innerHTML = xhr.responseText;

<?php
// 1. Simuler un délai de traitement
sleep(1);

// 2. Générer des données aléatoires
$donnees = array(
 "nom" => rand(1, 100),
 "texte" => "Texte aléatoire " . rand(1000, 9999)
);

// 3. Configurer l'en-tête pour indiquer que la réponse est en JSON
header('Content-Type: application/json');

// 4. Encoder les données en JSON et les envoyer
echo json_encode($donnees);
?>
```

on submit permet de vérifier le fichier avant la soumission :

```
<form onsubmit="return validateForm()">
 <input type="text" id="email" name="email">
 <input type="submit" value="Envoyer">
</form>

<script>
function validateForm() {
 var email = document.getElementById('email').value;
 var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

 if (!emailRegex.test(email)) {
 alert("Veuillez entrer une adresse email valide");
 return false; // Empêche la soumission
 }

 return true; // Permet la soumission
}
</script>
```

POST dans AJAX : Utilisé pour envoyer des données au serveur car il peut modifier l'état du serveur. Les données sont envoyées dans le corps de la requête HTTP idéal pour les données sensibles

```
<!DOCTYPE html>
<html lang="fr">
<head>
 <meta charset="UTF-8">
 <title>Ajouter un Utilisateur - Exemple AJAX POST</title>
 <script src="script.js"></script>
</head>
<body>
 <h1>Ajouter un Utilisateur</h1>
 <form id="userForm">
 <input type="text" id="nom" name="nom" placeholder="Nom" required>
 <input type="email" id="email" name="email" placeholder="Email" required>
 <input type="submit" value="Ajouter">
 </form>
 <div id="resultat"></div>
</body>
</html>

// Définition de la fonction qui va gérer notre requête AJAX
function ajouterUtilisateur() {
 // 1. Création d'un objet XMLHttpRequest
 var xhr = new XMLHttpRequest();

 // 2. Configuration de la fonction de callback
 xhr.onreadystatechange = function() {
 // 5. Vérification de l'état de la requête
 if (xhr.readyState === 4) { // 4 signifie que la requête est terminée
 if (xhr.status == 200) { // 200 est le code HTTP pour "OK"
 // 6a. La requête a réussi : on met à jour le contenu de la page
 document.getElementById('resultat').innerHTML = xhr.responseText;
 // 7. Réinitialiser le formulaire après une soumission réussie
 document.getElementById('userForm').reset();
 } else {
 // 6b. La requête a échoué : on affiche un message d'erreur
 document.getElementById('resultat').innerHTML = "Erreur : " + xhr.status;
 }
 }
 };

 // 3. Préparation des données et ouverture de la connexion
 var nom = document.getElementById('nom').value;
 var email = document.getElementById('email').value;
 var params = 'nom=' + encodeURIComponent(nom) + '&email=' + encodeURIComponent(email);
 xhr.open('POST', 'ajouter_utilisateur.php', true);

 // 4. Configuration des en-têtes et envoi de la requête
 xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
 xhr.send(params);

 // Configuration de l'écouteur d'événements une fois le DOM chargé
 document.addEventListener('DOMContentLoaded', function() {
 document.getElementById('userForm').addEventListener('submit', function(e) {
 e.preventDefault();
 ajouterUtilisateur();
 });
 });
}
```

```

<?php
header('Content-Type: application/json');

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
 $nom = isset($_POST['nom']) ? htmlspecialchars($_POST['nom']) : '';
 $email = isset($_POST['email']) ? htmlspecialchars($_POST['email']) : '';

 // Ici, vous ajouteriez normalement l'utilisateur à une base de données
 // Pour cet exemple, nous allons simplement simuler cela

 $success = true; // Simulons une opération réussie

 if ($success) {
 $response = [
 'status' => 'success',
 'message' => "Utilisateur $nom ajouté avec succès !",
 'user' => ['nom' => $nom, 'email' => $email]
];
 } else {
 $response = [
 'status' => 'error',
 'message' => "Erreur lors de l'ajout de l'utilisateur."
];
 }

 echo json_encode($response);
} else {
 echo json_encode(['status' => 'error', 'message' => 'Méthode non autorisée']);
}
?>

```

#### Les Promesses :

- Une promesse est un objet qui représente l'achèvement ou l'échec éventuel d'une opération asynchrone.

**États d'une promesse** :- En attente (pending) - Accomplie (fulfilled) - Rejetée les 3 choses qu'on peut mettre dans une promesse intermédiaire :

**Une valeur résolue - nouvelle promesse - Une erreur ou un rejet**

```

let promise = new Promise((resolve, reject) => {
 resolve(42);
});

promise.then(value => {
 return value + 1;
}).then(newValue => {
 console.log(newValue); // Affichera 43
});

let promise = new Promise((resolve, reject) => {
 resolve(42);
});

promise.then(value => {
 return new Promise((resolve, reject) => {
 setTimeout(() => {
 resolve(value + 1);
 }, 1000);
 });
}).then(newValue => {
 console.log(newValue); // Affichera 43 après 1 seconde
});

let promise = new Promise((resolve, reject) => {
 resolve(42);
});

promise.then(value => {
 if (value > 40) {
 return Promise.reject("Valeur trop élevée");
 }
 return value + 1;
}).then(newValue => {
 console.log(newValue);
}).catch(error => {
 console.error(error); // Affichera "Valeur trop élevée"
});

```

```

maPromesse
 .then((resultat) => {
 console.log(resultat); // "Succès !"
 })
 .catch((erreur) => {
 console.error(erreur); // "Échec !"
 });

```

**Promise.all** : tenue si toutes les promesses sont tenues.

**Promise.race** : tenue dès qu'une promesse est tenue.

```

function simulerRequete(url) {
 return new Promise((resolve, reject) => {
 console.log("État initial : en attente");
 setTimeout(() => {
 if (url === "https://api.example.com/success") {
 resolve({ id: 1, name: "Succès" });
 } else if (url === "https://api.example.com/error") {
 reject("Erreur : URL invalide");
 } else {
 reject("Erreur : URL inconnue");
 }
 }, 2000); // Simule un délai de 2 secondes
 });
}

// Cas 1 : Promesse accomplie (fulfilled)
console.log("Cas 1 : Promesse qui va réussir");
simulerRequete("https://api.example.com/success")
 .then(data => {
 console.log("Promesse accomplie. Données reçues :", data);
 })
 .catch(erreur => {
 console.error("Cette ligne ne devrait pas s'exécuter");
 })
 .finally(() => {
 console.log("Opération terminée (succès)");
 });

// Cas 2 : Promesse rejetée (rejected)
console.log("Cas 2 : Promesse qui va échouer");
simulerRequete("https://api.example.com/error")
 .then(data => {
 console.error("Cette ligne ne devrait pas s'exécuter");
 })
 .catch(erreur => {
 console.error("Promesse rejetée. Erreur :", erreur);
 })
 .finally(() => {
 console.log("Opération terminée (échec)");
 });

// Cas 3 : Promesse en attente (pending)
console.log("Cas 3 : Promesse en attente");
const promesseEnAttente = simulerRequete("https://api.example.com/unknown");
console.log("État immédiat de la promesse :", promesseEnAttente);

promesseEnAttente
 .then(data => {
 promesseEnAttente
 .then(data => {
 console.error("Cette ligne ne devrait pas s'exécuter");
 })
 .catch(erreur => {
 console.error("Promesse rejetée après attente. Erreur :", erreur);
 })
 .finally(() => {
 console.log("Opération terminée (après attente)");
 });
 });

```

Fetch est une API moderne en JavaScript pour effectuer des requêtes HTTP, il est basé sur les Promesses. Elle permet de récupérer des ressources, envoyer des données à un serveur, et traiter les réponses de manière asynchrone en utilisant des promesses. Plus simple à utiliser que XMLHttpRequest.

```

fetch('https://api.example.com/data', {
 method: 'POST',
 headers: {
 'Content-Type': 'application/json',
 },
 body: JSON.stringify({ key: 'value' })
})
 .then(response => {
 if (!response.ok) {
 throw new Error('Erreur réseau');
 }
 return response.json();
 })
 .then(data => {
 console.log('Succès:', data);
 })
 .catch(error => {
 console.error('Erreur:', error);
 });

```

Par défaut, Fetch n'envoie pas de cookies. Pour les inclure, utilisez { credentials: 'include' }.

### fetch-json.html

```
<html>
<head>
<script type="text/javascript" src="ajax.js"></script>
<script type="text/javascript" src="fetch-json.js"></script>
</head>
<body>
<p id="tofill"></p>
</body>
</html>
```

### fetch-json.js

```
window.addEventListener("load", function() {
 fetch("json.php", {method: "GET"})
 .then(response => response.json())
 .then(function(data) { alert(JSON.stringify(data)); })
 .catch(function(error) { alert("error " + error); });
});
```

### json.php

```
<?php
if (isset($_GET['source'])) die(highlight_file(__FILE__, 1));

header('Content-Type: application/json; charset: UTF-8');
$persion = array('nom' => 'Tremblay',
 'prenom' => 'Jean',
 'info' => array('age' => 40,
 'sexe' => 'masculin'
)
);

echo json_encode($persion);
?>
```

### promise-img.html

```
<html>
<head>
<script type="text/javascript" src="ajax.js"></script>
<script type="text/javascript" src="promise-img.js" defer></script>
</head>
<body>

</body>
</html>
```

### promise-img.js(exemple1)

```
// 1. Sélectionner l'élément image dans le DOM
const image = document.getElementById('fetch-image');

// 2. Faire une requête Fetch pour récupérer l'image
fetch("images/tintin.png")
 // 3. Traiter la réponse initiale
 .then(response => {
 if (response.ok) {
 // 4. Si la réponse est OK, convertir le contenu en blob
 return response.blob();
 }
 // 5. Si la réponse n'est pas OK, lancer une erreur
 throw Error(response.statusText);
 })
 // 6. Traiter le blob
 .then(blob => {
 // 7. Créer une URL d'objet à partir du blob
 const objectURL = URL.createObjectURL(blob);
 console.log("URL : " + objectURL);
 // 8. Définir cette URL comme source de l'image
 image.setAttribute("src", objectURL);
 })
 // 9. Gérer les erreurs potentielles
 .catch(err => console.log(err.message));
```

```
const image = document.getElementById('fetch-image');
const fonsc = function(response){
if (response.ok) {
return response.blob();
}
throw Error(response.statusText);
}
// c'est le premier qui se charge, qui s'affiche
Promise.race([fetch("images/gaston.png").then(fonsc), fetch("images/asterix.png")
.then(fonsc), fetch("images/tintin.png").then(fonsc)])
.blobs => {
const objectURL = URL.createObjectURL(blob);
console.log("URL : " + objectURL);
image.setAttribute("src", objectURL);
}
.catch(err => console.log(err.message));
```

```
const fonsc = function(response){
if (response.ok) {
return response.blob();
}
throw Error(response.statusText);
}
// Un tableau de promesses est renvoyé lorsque toutes les promesses sont tenues.
// Parcours avec forEach.
Promise.all([fetch("images/gaston.png").then(fonsc), fetch("images/asterix.png")
.then(fonsc), fetch("images/tintin.png").then(fonsc)])
.then(blobs => {
blobs.forEach(
blob, i) {
const image = document.getElementById('img' + i);
const objectURL = URL.createObjectURL(blob);
console.log("URL : " + objectURL);
image.setAttribute("src", objectURL);
}
})
.catch(err => console.log(err.message));
```

**async** et **await** sont des fonctionnalités de JavaScript qui simplifient la gestion des opérations asynchrones, rendant le code plus lisible et facile à écrire par rapport aux chaînes de promesses. Ils permettent de travailler avec des promesses de manière plus synchronisée, ce qui facilite la gestion des erreurs et la lecture du flux de code.

Avec les promesses, la gestion des erreurs peut devenir compliquée lorsqu'il y a plusieurs niveaux de promesses imbriquées. **async** et **await** permettent d'utiliser **try / catch** pour gérer les erreurs de manière similaire au code synchronisé.

Note : Await ne peut jamais se trouver à l'extérieur d'une fonction asynchrone.

```
// Fonction simulant une requête API qui retourne une promesse
function simulerRequeteAPI(id) {
 return new Promise((resolve, reject) => {
 setTimeout(() => {
 if (id > 0) {
 resolve({ id: id, nom: 'Utilisateur #' + id });
 } else {
 reject("ID invalide");
 }
 }, 1000); // Simule un délai d'une seconde
 });
}

// Fonction asynchrone principale
async function obtenirEtAfficherUtilisateur(id) {
 try {
 // Étape 1 : Début de la fonction asynchrone
 console.log("Début de la recherche de l'utilisateur");

 // Étape 2 : Attente de la résolution de la promesse
 // 'await' suspend l'exécution jusqu'à ce que la promesse soit résolue
 const utilisateur = await simulerRequeteAPI(id);

 // Étape 3 : Traitement du résultat
 // Cette ligne ne s'exécute que lorsque la promesse est résolue
 console.log("Utilisateur trouvé :", utilisateur);
 } catch (erreur) {
 // Étape 4 : Gestion des erreurs
 // Si une erreur se produit dans le bloc try, elle est capturée ici
 console.error("Une erreur s'est produite :", erreur);
 } finally {
 // Étape 5 : Nettoyage (optionnel)
 // Ce bloc s'exécute toujours, qu'il y ait eu une erreur ou non
 console.log("Recherche terminée");
 }
}

// Appel de la fonction asynchrone
obtenirEtAfficherUtilisateur(1);

function avecTimeout(promesse, delai) {
 let timeoutId;
 const timeoutPromise = new Promise((_, reject) => {
 timeoutId = setTimeout(() => {
 reject(new Error('Opération expirée'));
 }, delai);
 });

 return Promise.race([promesse, timeoutPromise])
 .finally(() => clearTimeout(timeoutId));
}

async function fonctionAvecTimeout() {
 try {
 const resultat = await avecTimeout(simulerRequeteAPI(1), 5000);
 console.log(resultat);
 } catch (erreur) {
 console.error('Erreur ou timeout:', erreur.message);
 }
}
```

Puisque Fetch retourne une Promesse, il peut effectivement être utilisé très efficacement avec Async/Await.

```

async function obtenerDonnees() {
 try {
 const response = await fetch('https://api.example.com/data');
 if (!response.ok) {
 throw new Error('Erreur réseau');
 }
 const data = await response.json();
 console.log(data);
 } catch (erreur) {
 console.error('Il y a eu un problème avec l\'opération fetch : ', erreur);
 }
}

obtenirDonnees();

```

CORS et JSONP sont utilisés pour permettre le partage de ressources entre différentes origines. CORS est un standard web moderne, fonctionne avec tous types de requêtes HTTP et offre un contrôle plus fin et sécurisé des accès. JSONP est une technique plus ancienne, fonctionne uniquement avec les requêtes GET et est moins sécurisé car il exécute du code JavaScript provenant d'un autre domaine.

**JSONP JSON with Padding** est une technique utilisée pour contourner la politique de sécurité de même origine (Same-Origin Policy) des navigateurs web lors de requêtes AJAX. Voici les points clés à retenir :

**1. Problème :** La politique de même origine empêche normalement les requêtes AJAX vers des domaines différents de celui de la page web.

**2. Solution :** JSONP utilise une balise <script> pour charger des données JSON depuis un autre domaine.

### 3. Fonctionnement :

- Le serveur enveloppe les données JSON dans une fonction callback.

- Le client définit cette fonction callback qui sera exécutée avec les données.

**4. Avantages :** Permet des requêtes cross-domain sans nécessiter CORS.

**5. Inconvénients :** Moins sécurisé que d'autres méthodes modernes, car il exécute du code provenant d'un serveur tiers.

```

<html lang="en">
<head>
 <meta charset="utf-8">
 <title>JSONP demo simple</title>

 <!-- Définition de la fonction callback pour JSONP -->
 <script type="application/javascript">
 // Cette fonction sera appellée avec les données JSON de Flickr
 var parseResponse = function (data) {
 // Extraction de l'URL de la première image du flux Flickr
 var d = data.items[0].media;
 // Redirection de la page vers l'URL de l'image
 window.location = d;
 };
 </script>

 <!-- Appel à l'API Flickr en utilisant JSONP -->
 <script type="application/javascript"
 src="http://api.flickr.com/services/feeds/photos_public.gne?format=json&jsoncallback=parseResponse">
 </script>
<!--
Explanation de l'URL de l'API Flickr :
- format=json : demande les données au format JSON
- jsoncallback=parseResponse : spécifie notre fonction callback

Fonctionnement :
1. Cette balise script fait une requête à l'API Flickr
2. Flickr renvoie les données JSON enveloppées dans un appel à parseResponse()
3. Le navigateur exécute le code JSON reçu, appelant ainsi notre fonction parseResponse()
4. parseResponse extrait l'URL de la première image et redirige la page
-->
</head>
<body>
 Ah Ah !
 <!--
 Note : Ce texte ne sera visible que brièvement avant la redirection.
 Si la redirection fonctionne, l'utilisateur verra rapidement une image Flickr à la place
 -->
</body>
</html>

```

Le paramètre `jsoncallback=parseResponse` est crucial ici, c'est le callback qui permet l'appel de la fonction qui fonctionne malgré l'appel à un script externe. Le contenu ressemble à : `parseResponse({ /* données JSON */ })`.

**Hors-contexte :** Alors, `<?php include("titre_page.html"); ?>` sera remplacé par ce contenu dans la page finale.

**CORS** est un mécanisme de sécurité implémenté par les navigateurs web pour contrôler les requêtes effectuées entre différentes origines. Décomposons cela :

**1. Origine (Origin) :** Une "origine" est définie par la combinaison du protocole (http, https), du domaine ([exemple.com](http://exemple.com)) et du port (80, 443, etc.). Par exemple, <https://exemple.com:443> est une origine.

**2. Politique de même origine (Same-Origin Policy) :** Par défaut, les navigateurs appliquent une politique de même origine, ce qui signifie qu'une page web ne peut faire des requêtes AJAX que vers sa propre origine.

### 3. Fonctionnement de CORS :

- Le navigateur envoie une requête avec un en-tête "Origin".

- Le serveur répond avec des en-têtes CORS spécifiques.

- Le navigateur vérifie ces en-têtes pour décider s'il autorise la requête.

### 4. Principaux en-têtes CORS :

- Access-Control-Allow-Origin : Spécifie quelles origines sont autorisées.

- Access-Control-Allow-Methods : Méthodes HTTP autorisées (GET, POST, etc.).

- Access-Control-Allow-Headers : En-têtes personnalisés autorisés.

### 5. Types de requêtes CORS :

- Simples : GET, POST, HEAD avec certains en-têtes standard.

- Réfléchies : Nécessitent une requête OPTIONS préalable pour vérifier les autorisations.

```

var invocation = new XMLHttpRequest();
var url = 'http://truc.autre/resources/public-data/';

function callOtherDomain() {
 if(invocation) {
 invocation.open('GET', url, true);
 // requête préliminaire
 invocation.setRequestHeader('Content-Type', 'application/xml');
 invocation.onreadystatechange = handler;
 invocation.send();
 }
}

var invocation = new XMLHttpRequest();
var url = 'http://truc.autre/resources/public-data/';

function callOtherDomain() {
 if (invocation) {
 invocation.open('GET', url, true);
 invocation.setRequestHeader('Content-Type', 'application/xml');
 invocation.onreadystatechange = function() {
 if (invocation.readyState === 4) {
 if (invocation.status === 200) {
 console.log('Success:', invocation.responseText);
 } else {
 console.error('Error:', invocation.statusText);
 }
 }
 };
 invocation.send();
 }
}

fetch('http://truc.autre/resources/public-data/', {
 method: 'GET',
 headers: {
 'Content-Type': 'application/xml'
 },
 // credentials: 'include' // Utilisez cette option si vous devez envoyer des informations sensibles
}).then(response => {
 if (!response.ok) {
 throw new Error('Network response was not ok ' + response.statusText);
 }
 return response.text(); // ou response.json() selon le type de réponse
})
.then(data => {
 console.log('Success:', data);
})
.catch(error => {
 console.error('Error:', error);
});

```

### Node-proxy :

Un proxy Node.js est un serveur intermédiaire qui agit comme un relais pour les requêtes entre un client (comme un navigateur web) et un serveur cible.

Fonctionnement pour contourner le cross-origin :

Principe de base : Au lieu de faire une requête directe du client vers un serveur d'une origine différente, le client fait une requête à son propre serveur (le proxy).

qui à son tour fait la requête au serveur cible.

```
// Importation des modules nécessaires
const express = require('express'); // Framework web pour Node.js
const morgan = require('morgan'); // Middleware de logging HTTP
// Middleware pour créer un proxy HTTP
const { createProxyMiddleware } = require('http-proxy-middleware');

// Création d'une instance d'application Express
const app = express();

// Configuration des constantes
const PORT = 3000; // Port sur lequel le serveur va écouter
const HOST = "localhost"; // Hôte sur lequel le serveur va écouter
const API_SERVICE_URL = "https://api.conceptnet.io/"; // URL de l'API cible

// Configuration du logging
// 'dev' est un format prédefini qui affiche des logs colorés et concis
app.use(morgan('dev'));

// Configuration du proxy pour les requêtes vers /conceptnet
app.use('/conceptnet', createProxyMiddleware({
 target: API_SERVICE_URL, // L'URL de l'API vers laquelle les requêtes seront redirigées
 changeOrigin: true, // Change l'origine de la requête vers la cible (important)
 pathRewrite: {
 '^/conceptnet': '', // Réécrit le chemin en supprimant '/conceptnet' de l'URL
 }
}));

// Démarrage du serveur
app.listen(PORT, HOST, () => {
 console.log(`Starting Proxy at ${HOST}:${PORT}`); // Log un message quand le serveur démarre
});
```

## React

```
const element = <h1>Bonjour, {nom}</h1>;
```

### Introduction

```
const root = ReactDOM.createRoot(document.getElementById('root'));
// Avec JSX
root.render(<h1>I Love JSX!</h1>);
```

ReactDOM.createRoot(document.getElementById('root')), vous dites à React : "Voici l'élément du DOM à partir duquel tu vas gérer et rendre toute l'interface utilisateur de l'application".

1. Isolation : Ce "root" définit les limites de votre application React. Tout ce qui est à l'intérieur sera géré par React, tandis que le reste de la page peut être géré par d'autres technologies si nécessaire.

2. Contrôle total : À partir de ce point d'ancre, React a le contrôle total. Il peut ajouter, modifier ou supprimer des éléments enfants de manière efficace et optimisée.

#### 1. Car.js :

```
function Car(props) {
 return <h2>I am a {props.color} Car!</h2>;
}
export default Car;
```

#### 1. useCar.js :

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Car from './Car.js';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<Car color="red" />);
```

import React from 'react'; importe la bibliothèque React (nécessaire même si non utilisé directement ici).  
- import ReactDOM from 'react-dom/client'; importe les fonctionnalités de rendu de React.  
- import Car from './Car.js'; importe le composant Car défini dans le fichier Car.js.  
- root.render(<Car color="red" />); rend le composant Car dans le DOM, en lui passant la prop color avec la valeur "red".

```
function sleep(ms) {
 return new Promise(resolve => setTimeout(resolve, ms));
}

</script>

<script type="text/babel">

function MissedGoal() {
 return <h1>MISSSED!</h1>;
}

function MadeGoal() {
 return <h1>Goal!</h1>;
}

// ----- condition type 1 -----
function Goal(props) {
 if (props.isGoal) {
 return <MadeGoal/>;
 }
 return <MissedGoal/>;
}

// ----- condition type 2 -----
function Goal(props) {
 return (
 <>
 { props.isGoal ? <MadeGoal/> : <MissedGoal/> }
 </>
);
}

// ----- condition type 3 -----
function Goal(props) {
 return (
 <>
 { props.isGoal && <MadeGoal/> }
 </>
);
}

// ----- fin condition -----
const root = ReactDOM.createRoot(document.getElementById('target'));
root.render(<Goal isGoal={true} />);

sleep(2000).then(() => { root.render(<Goal isGoal={false} />); });

</script>

<body>
 <!-- Div avec l'ID 'root' où notre application React sera rendue -->
 <div id="root"></div>

 <script type="text/babel">
 // Composant Car : affiche une marque de voiture dans un élément de liste
 function Car(props) {
 return I am a { props.brand };
 }

 // Composant Garage : contient une liste de voitures
 // Reçoit les voitures comme prop au lieu de les définir en interne
 function Garage(props) {
 return (
 // Fragment React (<> </>) pour regrouper plusieurs éléments sans div supplémentaire
 <>
 <h1>Who lives in my garage?</h1>

 /* Utilisation de map pour créer un composant Car pour chaque marque */
 /* Les voitures sont maintenant accessibles via props.cars */
 {props.cars.map((car) => <Car brand={car} />)}

 </>
);
 }

 // Création du point d'ancre React dans le DOM
 const root = ReactDOM.createRoot(document.getElementById('root'));

 // Définition du tableau de voitures à l'extérieur des composants
 const cars = ['Ford', 'BMW', 'Audi', 'Fiat'];

 // Rendu du composant Garage dans le point d'ancre
 // On passe le tableau cars comme prop au composant Garage
 root.render(<Garage cars={cars} />);
 </script>
</body>
```

```
// Composant Car : affiche une marque de voiture dans un élément de liste
function Car(props) {
 return I am a { props.brand };
}

// Composant Garage : contient une liste de voitures
function Garage() {
 // Tableau d'objets représentant les voitures
 // Chaque voiture a un id unique et une marque
 const cars = [
 {id: 1, brand: 'Ford'},
 {id: 2, brand: 'BMW'},
 {id: 3, brand: 'Audi'}
];

 return (
 // Fragment React (<> </>) pour regrouper plusieurs éléments sans div supplémentaire
 <>
 <h1>Who lives in my garage?</h1>

 /* Utilisation de map pour créer un composant Car pour chaque voiture */
 {cars.map((car) =>
 <Car
 key={car.id} // Propriété 'key' pour l'optimisation de React
 brand={car.brand} // Passage de la marque comme prop
 />
)}
 // ----- ou -----
 {(props.cars.map((car) => <Car key={car} brand={car} />))

 </>
);
}

// Crédit du point d'ancre React dans le DOM
const root = ReactDOM.createRoot(document.getElementById('root'));

// Rendu du composant Garage dans le point d'ancre
root.render(<Garage />);
```

#### Formulaire

```
<body>
 <!-- Div avec l'ID 'root' où notre application React sera rendue -->
 <div id="root"></div>

 <script type="text/babel">
 // Définition du composant MyForm en utilisant une fonction fléchée
 // Ce composant reçoit des props comme argument
 const MyForm = (props) => {
 return (
 // Rendu d'un formulaire HTML
 // Les attributs action et method sont définis dynamiquement via les props
 <form action={props.action} method={props.method}>
 /* Label et input pour le nom */
 <label>Enter your name:
 /* Input de type texte */
 /* Note : Cet input n'a pas de propriété 'name' ni de gestion d'état (*/
 <input type="text" />
 </label>
 </form>
);
 };

 // Crédit du point d'ancre React dans le DOM
 const root = ReactDOM.createRoot(document.getElementById('root'));

 // Rendu du composant MyForm
 // On passe les props 'action' et 'method' au composant
 root.render(<MyForm action="doIt.php" method="get" />);
 </script>
</body>
```

#### Hooks

```
// Utilisation du hook useState pour gérer l'état du composant
// 'name' est la variable d'état, 'setName' est la fonction pour la mettre à jour
const [name, setName] = React.useState("");

// Définition de la fonction handleSubmit pour gérer la soumission du formulaire
const handleSubmit = (event) => {
 // Empêche le comportement par défaut du formulaire (rechargement de la page)
 event.preventDefault();
 // Affiche une alerte avec le nom entré
 alert(`The name you entered was: ${name}`);
};

return (
 // Le formulaire utilise handleSubmit lors de sa soumission
 <form onSubmit={handleSubmit}>
 <label>Enter your name:
 <input
 type="text"
 // La valeur de l'input est contrôlée par l'état 'name'
 value={name}
 // Mise à jour de l'état 'name' à chaque changement de l'input
 onChange={(e) => setName(e.target.value)}
 />
 </label>
 </form>
);

// Crédit du point d'ancre React et rendu du composant MyForm
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

#### 2. Types de hooks courants :

- useState : pour ajouter l'état local à des composants fonctionnels.
- useEffect : pour effectuer des effets de bord (comme des appels API).
- useContext : pour accéder au contexte React.
- useReducer : pour gérer des états complexes.
- useRef : pour créer une référence mutable.

```
<!DOCTYPE html>
<html lang="fr">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Document</title>
</head>
<body>
 <div id="root"></div>
</body>
</html>
```

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import './styles.css';
import App from './App';
```

```
export default function Square() {
 return <button className="square">X</button>;
}
```

```
export default function Square() {
 return (
 <>
 <div className="board-row">
 <button className="square">1</button>
 <button className="square">2</button>
 <button className="square">3</button>
 </div>
 <div className="board-row">
 <button className="square">4</button>
 <button className="square">5</button>
 <button className="square">6</button>
 </div>
 <div className="board-row">
 <button className="square">7</button>
 <button className="square">8</button>
 <button className="square">9</button>
 </div>
 </>
);
}
```

```
function Square({ value }) {
 return <button className="square">{value}</button>;
}
export default function Board() {
 return (
 <>
 <div className="board-row">
 <Square value="1" />
 <Square value="2" />
 <Square value="3" />
 </div>
 <div className="board-row">
 <Square value="4" />
 <Square value="5" />
 <Square value="6" />
 </div>
 <div className="board-row">
 <Square value="7" />
 <Square value="8" />
 <Square value="9" />
 </div>
 </>
);
}
```

```

function Square({ value }) {
 function handleClick() {
 console.log('cliqué !');
 }

 return (
 <button
 className="square"
 onClick={handleClick}
 >
 {value}
 </button>
);
}

```

Les "props" (abréviation de "properties") sont un concept fondamental en React. Ce sont des données qu'un composant parent peut passer à ses composants enfants. Elles permettent de rendre les composants plus dynamiques et réutilisables.

```

export default function Board() {
 const [squares, setSquares] = useState(Array(9).fill(null));
 return (
 <>
 <div className="board-row">
 <Square value={squares[0]} />
 <Square value={squares[1]} />
 <Square value={squares[2]} />
 </div>
 <div className="board-row">
 <Square value={squares[3]} />
 <Square value={squares[4]} />
 <Square value={squares[5]} />
 </div>
 <div className="board-row">
 <Square value={squares[6]} />
 <Square value={squares[7]} />
 <Square value={squares[8]} />
 </div>
 </>
);
}

import { useState } from 'react';
//2.Ajoutez ensuite la fonction onSquareClick aux props du composant Square
function Square({ value, onSquareClick }) {
 return (
 //1. Commencez par définir la fonction que le composant Square appellera lors du clic.
 // nommé onsquareclick
 <button className="square" onClick={onSquareClick}>
 {value}
 </button>
);
}

//4.us définirez la fonction handleClick au sein du composant Board
// pour qu'elle mette à jour le tableau squares représentant l'état de votre plateau

//note : La fonction handleClick crée une copie du tableau squares (nextSquares)
// grâce à la méthode de tableau JavaScript slice().
// Ensuite, handleClick met à jour le tableau nextSquares pour ajouter
// un X à la première case (index [0]).

//On appelle alors la fonction setSquares pour avertir React que l'état du composant
// a changé. Ça déclenchera un nouvel affichage des composants qui utilisent
// l'état squares (donc Board), ainsi que de tous leurs composants enfants
// (les composants Square qui constituent le plateau).

//5.Ajoutez un paramètre i à la fonction handleClick,
// destiné à recevoir l'index de la case du plateau à modifier
return (
 <>
 <div className="board-row">
 <Square value={squares[0]} onSquareClick={() => handleClick(0)} />
 <Square value={squares[1]} onSquareClick={() => handleClick(1)} />
 <Square value={squares[2]} onSquareClick={() => handleClick(2)} />
 </div>
 <div className="board-row">
 <Square value={squares[3]} onSquareClick={() => handleClick(3)} />
 <Square value={squares[4]} onSquareClick={() => handleClick(4)} />
 <Square value={squares[5]} onSquareClick={() => handleClick(5)} />
 </div>
 <div className="board-row">
 <Square value={squares[6]} onSquareClick={() => handleClick(6)} />
 <Square value={squares[7]} onSquareClick={() => handleClick(7)} />
 <Square value={squares[8]} onSquareClick={() => handleClick(8)} />
 </div>
 </>
);
}

```

La communication entre enfant se fait aussi par le parent :

```

import React, { useState } from 'react';

function ChildA({ onMessageSend }) {
 return (
 <button onClick={() => onMessageSend("Hello from A")}>
 Send message to B
 </button>
);
}

function ChildB({ message }) {
 return <div>Message received: {message}</div>;
}

function Parent() {
 const [message, setMessage] = useState("");
 const handleMessageSend = (newMessage) => {
 setMessage(newMessage);
 };

 return (
 <div>
 <ChildA onMessageSend={handleMessageSend} />
 <ChildB message={message} />
 </div>
);
}

export default Parent;

import { useState } from 'react';

function Square({value, onSquareClick}) {
 return (
 <button className="square" onClick={onSquareClick}>
 {value}
 </button>
);
}

export default function Board() {
 const [xIsNext, setXIsNext] = useState(true);
 const [squares, setSquares] = useState(Array(9).fill(null));

 function handleClick(i) {
 if (squares[i]) {
 return;
 }
 const nextSquares = squares.slice();
 if (xIsNext) {
 nextSquares[i] = 'X';
 } else {
 nextSquares[i] = 'O';
 }
 setSquares(nextSquares);
 setXIsNext(!xIsNext);
 }

 return (
 <>
 <div className="board-row">
 <Square value={squares[0]} onSquareClick={() => handleClick(0)} />
 </div>
);
}

export default function Board() {
 //...
}

function calculateWinner(squares) {
 const lines = [
 [0, 1, 2],
 [3, 4, 5],
 [6, 7, 8],
 [0, 3, 6],
 [1, 4, 7],
 [2, 5, 8],
 [0, 4, 8],
 [2, 4, 6]
];
 for (let i = 0; i < lines.length; i++) {
 const [a, b, c] = lines[i];
 if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
 return squares[a];
 }
 }
 return null;
}

function handleClick(i) {
 if (squares[i] || calculateWinner(squares)) {
 return;
 }
}

```

```

export default function Board() {
 // ...
 const winner = calculateWinner(squares);
 let status;
 if (winner) {
 status = winner + " a gagné";
 } else {
 status = "Prochain tour : " + (xIsNext ? "X" : "O");
 }

 return (
 <>
 <div className="status">(status)</div>
 <div className="board-row">
 // ...
)
}

export default function Game() {
 const [xIsNext, setXIsNext] = useState(true);
 const [history, setHistory] = useState([Array(9).fill(null)]);
 const currentSquares = history[history.length - 1];

 function handlePlay(nextSquares) {
 setHistory([...history, nextSquares]);
 setXIsNext(!xIsNext);
 }

 return (
 <div className="game">
 <div className="game-board">
 <Board xIsNext={xIsNext} squares={currentSquares} onPlay={handlePlay} />
 </div>
 <div className="game-info">
 /*TODO
 </div>
 </div>
);
}

export default function Game() {
 const [xIsNext, setXIsNext] = useState(true);
 const [history, setHistory] = useState([Array(9).fill(null)]);
 const currentSquares = history[history.length - 1];

 function handlePlay(nextSquares) {
 setHistory([...history, nextSquares]);
 setXIsNext(!xIsNext);
 }

 function jumpTo(nextMove) {
 // TODO
 }

 const moves = history.map((squares, move) => {
 let description;
 if (move > 0) {
 description = 'Aller au coup #' + move;
 } else {
 description = 'Revenir au début';
 }
 return (

 <button onClick={() => jumpTo(move)}>(description)</button>

);
 });

 return (
 <div className="game">
 <div className="game-board">
 <Board xIsNext={xIsNext} squares={currentSquares} onPlay={handlePlay} />
 </div>
 <div className="game-info">
 (moves)
 </div>
 </div>
);
}

const moves = history.map((squares, move) => {
 // ...
 return (
 <li key={move}>
 <button onClick={() => jumpTo(move)}>(description)</button>

);
});

```

```

export default function Game() {
 const [xIsNext, setXIsNext] = useState(true);
 const [history, setHistory] = useState([Array(9).fill(null)]);
 const [currentMove, setCurrentMove] = useState(0);
 const currentSquares = history[history.length - 1];
 // ...

}

export default function Game() {
 // ...
 function jumpTo(nextMove) {
 setCurrentMove(nextMove);
 setXIsNext(nextMove % 2 === 0);
 }
 // ...

}

function handlePlay(nextSquares) {
 const nextHistory = [...history.slice(0, currentMove + 1), nextSquares];
 setHistory(nextHistory);
 setCurrentMove(nextHistory.length - 1);
 setXIsNext(!xIsNext);
}

```

Une API REST (Representational State Transfer) est un style d'architecture pour la conception d'applications en réseau, particulièrement utilisé pour créer des services web. Voici les points clés à comprendre :

1. API signifie "Application Programming Interface" : C'est un ensemble de règles et de protocoles qui permettent à différentes applications de communiquer entre elles.
2. REST est un style architectural : Il définit un ensemble de contraintes pour créer des services web scalables.
3. Basée sur le protocole HTTP : Une API REST utilise les méthodes HTTP standard (GET, POST, PUT, DELETE, etc.) pour effectuer des opérations sur les ressources.
4. Orientée ressource : Dans une API REST, tout est considéré comme une ressource, identifiée par une URL unique.
5. Sans état (Stateless) : Chaque requête du client au serveur doit contenir toutes les informations nécessaires pour comprendre et traiter la requête. Le serveur ne garde pas l'état du client entre les requêtes.
6. Représentations : Les ressources peuvent avoir différentes représentations, généralement JSON ou XML.
7. Opérations CRUD : Les API REST sont souvent utilisées pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur les ressources.

#### MERN :

NodeJS fournit aussi un gestionnaire de paquet pour l'installation de logiciel supplémentaires : npm.  
Express va gérer les routes, les connexions à la base de données, l'authentification,  
Les données ne sont pas stockées sous forme de tables ayant des relations mais sous forme de collections d'objets ce qui sera particulièrement pratique avec JS