Pour rendre vos pages dynamiques, on va utiliser JavaScript !
Il nous permettra d'interagir avec notre page HTML !

la doc : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

---

# Inclure le code JavaScript dans le HTML

Soit on met le code directement dans le HTML, entre les balises `<script>` :

```
<script>
        a = 1;
        b = a + 1;
        console.log(b);
</script>
```

on bien, on désigne à HTML où trouver notre fichier `mon-code.js` avec notre code JavaScript, dans l'attribut `src` de la balise `<script>` :

```
<script src="mon-code.js"></script>
```

> **Remarque**
>
> On peut placer l'élément `<script>` n'importe où dans le HTML.

---

# Syntaxe

Les instructions sont séparées par de point-virgules (comme en C).
Techniquement, ça marche très bien sans les point-virgules, mais c'est mieux avec !

## Les variables

### Les variables `let`

On déclare une variable avec le mot-clé `let` suivi du nom qu'on veut donner à la variable.
C'est le même mot-clé quel que soit le type de la variable.

```
let a = 1; // a est un entier

let f = 2.5; // f est un nombre flottant

let str = "chaine"; // str est une chaîne de caractères

let tab = [1, 2, 3] // tab est un array d'entier
```

## Les variables constantes `const`

Pour créer des constantes qu'on peut lire, mais jamais modifier on utilise le mot-clé `const` pour déclarer une variable constante.

Tant que le programme tourne, cette variable ne pourra pas être modifiée.

Elle doit être initialisée à sa déclaration.

```
const CST = 10;

CST = 11; // TypeError: invalid assignment to const 'number'

const CST_2; // SyntaxError: Missing initializer in const
```

> **Remarque**
>
> Par convention, on nomme une variable constante en majuscule.

## Portée des variables

Une variable déclarée avec `let` ou `const` n'existe que dans le bloc où elle a été déclarée.

```
let a = 1;

if (True) {
        let b = 2; // ce b n'existera plus en sortant du bloc IF
}

console.log("a = " + a); // a = 1
console.log("b = " + b); // ReferenceError: b is not defined
```

Plus d'info sur la portée de `let` et `const`.

> **Remarque**

Il existe une troisième façon de déclarer une variable avec le mot-clé `var` dont la portée est différente. Elle n'est plus utilisée, mais il se peut que vous tombiez sur du code avec `var` dedans. On n'en parlera pas dans ce cours, mais pour les curieux et les curieuses vous pouvez lire la doc de `var`.

### Déclarer plusieurs variables d'un coup

On peut déclarer plus d'une variable sur la même ligne :

```
let a = 1, b = 2, str = "chaine", tab = [1, 2];

const a2 = 1, b2 = 2, str = "chaine", tab2 = [1, 2];
```

## Les conditions

Rien de bien nouveau au niveau des conditions, on utilise l'instruction `if...else` comme en C.

```
let age = 22;

if (age > 18) {
        console.log("Je suis majeur.");
} else if (age == 18) {
        console.log("Je suis majeur cette année.");
} else {
        console.log("Je suis mineur.");
}
```

## Les boucles

### La boucle `for`

La boucle `for` a la même syntaxe qu'en C :

```
for (let i=0; i < 5; i++) {
        console.log(i);
}
```

Ce bout de code signifie qu'au début de la boucle on initialise la variable `i` à 0, et après avoir effectué toutes les instructions dans la boucle, on incrémente `i` de 1, et on teste la condition `i < 5`. Si la condition est respectée, on part sur un nouveau tour de boucle, sinon la boucle est

finie. Dans l'exemple, on a donc un compteur `i` qui aura les valeurs 0, puis 1, ..., jusqu'à 4 (donc 5 tours de boucle).

On pourrait aussi très bien compter à l'envers :

```
for (let i=5; i > 0; i--) {
        console.log(i); // 5, 4, 3, 2, 1
}
```

On fait aussi 5 tours de boucles, mais les valeurs que prend la variable `i` sont différentes.

### La boucle `while`

C'est une boucle `while` typique. Elle prend une condition, et tant qu'elle est vraie, on rerentre dans la boucle.

```
let i = 0;

while (i < 5) {
        console.log(i); // 0, 1, 2, 3, 4
        i++;
}
```

### La boucle `do while`

Dans un `while` "normal", on ne rentre pas dans la boucle si la condition n'est pas vérifiée dès le début.

La boucle `do...while` est une boucle qui fait en sorte que le code dans la boucle est exécuté au moins une fois. Quand le code a été exécuté cette première fois, on refait un tour de boucle uniquement si la condition est respectée.

```
let i = 0;

do {
        i += 1;
} while (i > 1);

console.log(i); // affiche 1
```

## Les fonctions

Il y a deux types de fonctions en JS : le fonctions "normales", et les fonctions dites anonymes.

## Les fonctions typiques

La fonction normale se déclare avec `function` suivi de son nom, et ses arguments entre parenthèses :

```
function nom_fonction(arg1, arg2) {
  return arg1 * arg2 * 3.5;
}
```

## Les fonctions anonymes

En JS, on a tendance à beaucoup utiliser les fonctions anonymes, c'est-à-dire simplement des fonction qui n'ont pas de nom. Il y a deux façon de les déclarer :

```
// fonction anonyme déclarée avec `function`
const f = function(a) {
  console.log(a + 2);
};
```

ou

```
// fonction anonyme avec une syntaxe plus réduite
let h = (a) => { console.log(a + 2); };
```

## Les tableaux

Les tableaux sont un des objets standards de JavaScript, c'est-à-dire qu'ils proposent de nombreuses méthodes qui permettent de les manipuler.

1. Créer un tableau

```
let tab = [1, 2, 3];
```

2. Accéder à une case du tableau :

```
console.log(tab[0]); // affiche 1
```

3. Connaître la taille du tableau avec `length` :

```
console.log(tab.length); // affiche 3
```

4. Boucle sur un tableau :

```
tab.forEach((element) => console.log(element)); // affiche 1 2 3
```

La méthode `forEach` prend une fonction qui a pour argument un element de tableau. Cette fonction sera appliquée pour chacun des éléments du tableau.

(Si la notation des cet exemple n'a pas de sens pour vous, retournez lire la partie sur les fonctions anonymes.

5. Ajouter un élément à la fin du tableau :

```
tab.push(4); // maintenant le tableau vaut [1, 2, 3, 4]
```

Il existe tout un tas d'autres méthodes, trop pour toutes les lister. Avant de vous lancer à essayer de coder votre propre fonction pour faire quelque chose sur un tableau, checkez d'abord la documentation des Array pour voir si un outil qui fait ce que vous voulez faire existe..

# Interagir avec le DOM

## Les éléments

la doc d'un élément

des propriétés intéressantes des éléments :

- `id` : retourne la valeur de l'attribut `id` de l'élément ;
- `className` : retourne la valeur de l'attribut `class` de l'élément ;
- `innerHTML` : retourne sous forme de chaîne de caractère le contenu de l'élément ;
- `getAttribute("nom-attribut")` : retourner la valeur de l'attribut entré en argument ;
- `style` : récupère les propriétés CSS dans l'attribut `style`

Ces propriétés sont accessibles en lecture et en écriture (par exemple, je peux changer l'`id` d'un élément depuis le JS).

## Récupérer et modifier un élément

Les méthodes de `Document` suivantes permettent de renvoyer un ou plusieurs `Element` :

- `getElementById` : retourne l'élément dont l'`id` est passé en argument ;
- `getElementsByClassName` : retourne l'ensemble des éléments dont la valeur de l'attribut `class` est passé en argument ;
- `getElementsByTagName` : retourne l'ensemble des éléments dont le nom de la balise est passé en argument ;

### Changer le style d'un élément

Disons que dans le corps de mon HTML, la balise suivante existe :

```
<h1 id="t1">Mon titre</h1>
```

alors je peux modifier le style en JS :

```
let h1 = document.getElementById("t1");
h1.style.color = "red";
h1.style.border = "solid 1px"
```

### Changer le contenu d'un élément

Toujours sur le HTML de l'exemple précédant :

```
let h1 = document.getElementById("t1");

h1.innerHTML = ""; // j'ai effacé tout le contenu de mon élément

h1.innerHTML = "Mon nouveau titre"; // maintenant, h1 a un nouveau contenu
```

On peut même écrire à la main du nouveau code HTML :

```
let div = document.getElementById("div1");

let str = "<ul>";

for (let i=0; i < 5; i++) de {
        str += "<li>" + i + "</li>";
}

str += "</ul>";

div.innerHTML = str;
```

Ce bout de code ajouter à la page une liste de 0 à 4, et on n'a pas eu besoin de l'écrire ligne par ligne.

## Soyez sympas avec la prof, écrivez du code propre :)

- donnez à vos variables des noms qui ont du sens ;
- indentez proprement votre code ;

- mettez des espaces aux bons endroits.

# Biblio


- https://eloquentjavascript.net/
- https://www.quirksmode.org/js/contents.html
- https://ensweb.users.info.unicaen.fr/TW4b/pres/js/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference