**Support de cours JS**

**Sommaire**

1- Déclaration de constantes et variables

2- Les types de données

3- Les chaines de caractères

3.1- Concaténation

3.2- La longueur

3.3- Fonctions de chaines de caractères

4- Les tableaux

5- Les objets

6- Les boucles

7- Les structures de contrôle

7.1- If/else

7.2- L’opérateur ternaire

7.3- Switch

8- Les fonctions

9- Le DOM

9.1- Récupérer des éléments du DOM

9.2- Manipilation du DOM

10- Les évènements

**JavaScript**

(Ne doit pas être confondu avec Java)

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives et à ce titre est une partie essentielle des applications web. Avec les technologies HTML et CSS, JavaScript est parfois considéré comme l'une des technologies cœur du World Wide Web. Une grande majorité des sites web l'utilisent, et la majorité des navigateurs web disposent d'un moteur JavaScript dédié pour l'interpréter, indépendamment des considérations de sécurité qui peuvent se poser le cas échéant.

On commence par la creation d’un squelette d’une page HTML

<!doctype html>

<html lang="fr">

<head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <title>Cours JS</title>

</head>

<body>

   <h1>Cours JS</h1>

</body>

</html>

On va ajouter une partie script pour ajouter les scripts js 🡺

<script>

</script>

**NB :** pour des raisons de performance, il est recommandé de mettre les balises <script> avant </body>

**// mise du js à l’intérieur du fichier HTML**  
alert ("Mon 1er cours JS");

**Explication** de la fonction alert() ;

**// Utilisation de js d’un fichier externe**

On crée le fichier main.js et on deplace dedans le contenu js (on ne copie pas les balises <script>)🡺on ajoute l’attribut src à la balise script pour spécifier le chemin du fichier externe 🡺 pour eviter de faire OK à chaque fois sur les alertes, on peut la console

console.log("Mon 1er cours JS");

**1- Déclaration de constantes et variables**

**explic** : constante et variable

**exemple :**

const i = 10; *// declaration et initialisation de constante*

**NB :** les constantes ne peuvent pas changer de valeur, sinon ca déclenche une erreur.

Pour vouloir utiliser une var dont la valeur change, on utilise le mot clé let.

🡺 changer const de age par let 🡺afficher une var dans la console

let adresse ;

let age = 30;

console.log(age);

age = 35;

console.log(age);

console.log("Mon age est : "+age); *// affichage soigné*

Le signe + fait la concaténation entre différentes variables et/ou chaines de caractères.

**2- Les types de données**

[[ commenter le bloc precedent ]]

const nom="Julien";

const age = 33;

const moyenne= 18.5;

const isCool=true;

const x=null;

const y=undefined;

**ex** : afficher le type de chacune des variables dans la console

pour connaitre le type d’une var, on utilise la fonction typeof

console.log(typeof age);

**3- Les chaines de caractères**

**3.1- Concaténation**

**ex** : afficher le message suivant dans la console

Ma note en JS est : x (x : est le contenu de la var moyenne)

console.log("Ma note en JS est : " + moyenne); // affichage classique

console.log(`Ma note en JS est : ${moyenne}`); // JS moderne ES6

**3.2- La longueur**

**ex** : afficher le nombre de caractères du nom

console.log(`Mon nom est composé de ${nom.length} caractères`);

**3.3- Fonctions de chaines de caractères**

console.log(`Mon nom en maj : ${nom.toUpperCase()}`); // nom en maj

console.log(`Mon nom en min : ${nom.toLowerCase()}`); // nom en min

console.log(`Sous chaine : ${nom.substring(0,4)}`); *// extrait une sous*

*chaine, les index de 0 à 3 ==> 4 caract*

**NB :** les index commenecent à 0

console.log(`Sous chaine : ${nom.substring(0,4).toUpperCase()}`); *// prend les 4 caract et les transforme en maj*

🡺 chainage de fonctions : appliquer une fonction à la suite de l’autre

console.log(nom.split('')); *// decoupe la chaine en caractères*

const nomComplet = "Julien La Garde";

console.log(nomComplet.split(' ')); *// decoupe la chaine selon un*

*séparateur (dans ce cas, c'est l'espace) 🡺 3 mots*

**ex** : mettre une phrase dans une chaine et afficher le nombre de mots qui la composent

**4- Les tableaux**

*// decla de tableaux*

const numbers = new Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

const fruits = ["mangue", "pastèque", "poire", "orange", "ananas"];

*// acces a un element du tableau*

console.log(fruits[0]); *// acces au 1ere element*

console.log(fruits[4]); *// acces au 5eme element*

fruits.unshift("fraise"); *// ajout au debut du tableau*

fruits.push("banane"); *// ajout en fin de tableau*

fruits.shift(); *// supprimer le 1er element*

fruits.pop(); *// supprimer le dernier element*

console.log(fruits.indexOf("orange")); *// retourne la position de l'element orange*

console.log(Array.isArray(fruits));  *// on teste si une var est de type tableau, dans cet exemple true*

liste complète des fonctions de tableaux

https://www.w3schools.com/jsref/jsref\_obj\_array.asp

**5- Les objets**

const personne = {

  prenom: "Julien",

  nom: "La garde",

  age: 30,

  passions: ["foot", "natation", "jeux videos"],

  adresse : {

    rue : "rue Nationale",

    ville : "Lille",

    pays : "France"

  }

};

console.log(personne); *//  afficher un objet*

*// acces aux attributs de l'objet*

console.log(personne.prenom); *// acces au prenom*

console.log(personne.passions[0]); *// acces au 1er element des passions*

console.log(personne.adresse.rue); *// acces a la rue (qui est lui meme un attribut de l'objet adresse')*

personne.prenom="Julie"; *// modifie la valeur d'un attribut*

console.log(personne.prenom); *// afficher la nouvelle valeur*

personne.email="julie@example.com"; *// ajout d'un nouvel attribut*

console.log(personne); *// affiche le nouvel attribut*

const todos = [

  {

    id: 1,

    text: "Faire les courses",

    isCompleted: true,

  },

  {

    id: 2,

    text: "Balade au vieux Lille",

    isCompleted: true,

  },

  {

    id: 3,

    text: "Préparer le diner",

    isCompleted: false,

  },

  {

    id: 4,

    text: "Regarder la TV",

    isCompleted: false,

  },

];

console.log(todos); *// afficher un tableau d'objets*

console.log(todos[0].text); *// afficher l'attribut text du 1er objet du*

*tableau*

**6- Les boucles**

for (let i = 0; i < todos.length; i++) {

  document.write(todos[i].text + "<br>");

}

document.write("<br><br>");

let i = 0;

while (i < todos.length ) {

  document.write(todos[i].text + "<br>");

  i++;

}

document.write("<br><br>");

todos.forEach(function (todo) {

  document.write(todo.text + "<br>");

});

document.write("<br><br>");

const listeTodo=todos.map(function (todo) {

return todo.text;

});

console.log(listeTodo); *// tableau qui contient la liste des textes des taches*

La méthode map() ne modifie pas le tableau d'origine, elle crée un nouveau tableau en exécutant une fonction de callback sur chaque élément du tableau.

const completed = todos.filter(function (todo) {

  return todo.isCompleted === true;

});

console.log(completed); *// tableau qui contient la liste des taches finies*

La méthode filter() ne modifie pas le tableau d'origine, elle crée un tableau rempli de tous les éléments du tableau qui réussissent un test (spécifié dans la fonction du callback).

On peut factoriser l’exemple precedent grace à la technique de chainage de fonctions

const completed = todos

  .filter(function (todo) {

    return todo.isCompleted === true;

  })

  .map(function (todo) {

    return todo.text;

  });

console.log(completed); *// tableau qui contient la liste des taches finies*

**7- Les structures de contrôle**

**7.1- If/else**

**Syntaxe générale :**

if (*condition1*) {  
 //  traitement si condition 1 est vraie} else if (*condition2*) {  
 //  traitement si condition 2 est vraie} else {

//  le reste des cas   
}

**NB :** les blocs, else et else if ne sont pas obligatoires, seule le bloc if l’est.

La condition qui suit le if est condition logique, elle sera évaluée à true ou à false .

**Exemple complet :**

const x = 14;

const y = 10;

if (x == y) {

  console.log("les deux nombres sont égaux");

} else if (x > y) {

  console.log("x est supérieur à y");

}

else {

  console.log("x est inférieur à y");

}

**NB :** on peut combiner les opérateurs de comparaison.

if (x > y && x < 20) {

  console.log("nombre autorisé");

}

**7.2- L’opérateur ternaire**

**Syntaxe**

nom\_variable = (condition) ? valeur si vrai : valeur si faux

**Exemple**

const message = x > 10 ? "commande permise" : "commande non permise";

console.log(message);

**7.3- Switch**

**Syntaxe**

switch(expression) {  
  case x:  
    *// traitement*    break;  
  case y:  
    *// traitement*

    break;  
  default:  
    // *traitement*  
}

**Exemple**

const age = 10;

switch (age) {

  case 18:

    console.log("majeur");

    break;

  case 60:

    console.log("retraité");

    break;

  default:

    console.log("autres");

}

**NB :** Les switch utilisent une comparaison stricte (===).  
Les valeurs doivent être du même type pour correspondre.

**Exemple 2**

switch (true) {

  case age > 0 && age <= 12:

    console.log("enfant");

    break;

  case age >= 13 && age < 18:

    console.log("adolescent");

    break;

  case age >= 18:

    console.log("majeur");

    break;

  default:

    console.log("erreur");

}

**NB :** Si les valeurs des cases sont des expressions et non pas des valeurs exactes, il faut mettre true dans la clause switch.

**8- Les fonctions**

Une fonction JavaScript est un bloc de code conçu pour effectuer une tâche particulière.  
Une fonction JavaScript est exécutée lorsque 'quelque chose' l'invoque (l'appelle).

**Syntaxe**

function nom de la fonction (paramètre1, paramètre2, …. , paramètre n) {  
// code à exécuter  
}

Activité 1 : essayer de décortiquer le fonctionnement des fonctions en solo

*// implementation d'une fonction*

*// fonction avec param et retour de valeur*

function somme(a, b) {

  return a + b;

}

*// fonction sans param et sans retour de valeur*

function affichageDate() {

  const date = new Date();

  document.write(

    `aujourd'hui il est le ${date.getDate()}/${date.getMonth()+1}/${date.getFullYear()}`

  );

}

x = 5;

y = 10;

*// appel des fonctions*

total = somme(x, y);

document.write(total);

document.write(somme(x, y));

document.write(somme(12, 20));

affichageDate();

**[[ Explication** **du code ]]**

**Fonctions fléchées**

Une expression de fonction fléchée (arrow function en anglais) permet d’avoir une syntaxe plus courte que les expressions de fonction. Les fonctions fléchées sont souvent anonymes et ne sont pas destinées à être utilisées pour déclarer des méthodes.

Les fonctions fléchées n’ont pas besoin du couple d’accolades classique aux fonctions pour fonctionner et n’ont pas besoin non plus d’une expression return puisque celles-ci vont automatiquement évaluer l’expression à droite du signe => et retourner son résultat.

**Syntaxe**

(param 1, param 2, … param n) => {

// instructions

}

**Exemple :**

const voitures = [

  'Ferrari',

  'Mercedes',

  'BMW',

  'Maserati'

];

console.log(voitures.map( (voiture) => voiture.length ) );

*// Array(4) [ 7, 8, 3, 8 ]*

Fonction classique vs fonction fléchée

*/\*Expression de fonction classique :*

let somme = function (a, b) {

  return a + b;

};

*\*/*

*//Equivalent en fonction fléchée : ES6*

let somme = (a, b) => a + b;

console.log(somme(1, 2));

**9- Le DOM**

**9.1- Récupérer des éléments du DOM**

const form=document.getElementById("my-form");

*// recuperer l'element qui a l'id my-form*

console.log(form);

const conteneurs = document.getElementsByClassName("container"); *// recuperer une collection d'elements qui ont la classe container*

console.log(conteneurs);

const inputName = document.getElementsByName("name"); *// recuperer une collection d'elements dont l’attribut name est spécifié en param*

const ul = document.getElementsByTagName("ul"); *// recuperer une collection d'elements qui ont la balise ul*

La méthode querySelector() renvoie le premier élément qui correspond à un ou plusieurs sélecteurs CSS spécifiés dans le document.  
NB: La méthode querySelector() ne renvoie que le premier élément qui correspond aux sélecteurs spécifiés. Pour renvoyer toutes les correspondances, utilisez plutôt la méthode querySelectorAll().

const sectionContainer = document.querySelector("section.container"); *// recuperer le 1ere element qui repond au selecteur CSS spécifié en param*

const allInputs = document.querySelectorAll("input"); *// recuperer une collection d'elements qui ont repondent au selecteur CSS spécifié en param*

**9.2- Manipilation du DOM**

document.querySelector(".para1").remove();  *// supprimer l'element*

**NB :** Commenter cette ligne apres l’avoir testée

console.log(document.querySelector(".para1").textContent);

*// recupérer le contenu textuel d'un element*

document.querySelector(".para1").textContent = "para modifié";

*// modifie le contenu textuel d'un element*

document.querySelector(".para1").innerText = "para modifié 2eme fois";

**NB :** Cette propriété est similaire à la propriété textContent, mais il existe quelques différences:  
- textContent renvoie le contenu textuel de tous les éléments, tandis que innerText renvoie le contenu de tous les éléments, à l'exception des éléments de script et de style.  
- innerText ne renverra pas le texte des éléments masqués avec CSS (textContent le fera).

document.querySelector(".para1").innerHTML = "<span class='p2'>para  de classe p2 </span>"; *// modifie le contenu html (on peut mettre des balises) d'un element*

Liste des attributs et méthodes du DOM

https://www.w3schools.com/jsref/dom\_obj\_document.asp

**10- Les évènements**

Considérons le fichier html suivant :

<!doctype html>

<html lang="fr">

<head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <title>Cours JS</title>

    <link rel="stylesheet" href="styles.css">

</head>

<body>

    <h1 class="titre">Cours JS</h1>

    <p class="para1">Lorem ipsum dolor sit, amet consectetur adipisicing elit. Illo maxime explicabo cupiditate ipsam

        autem. Estnecessitatibus, architecto esse quaerat hic repellat libero nisi ad eos ex vero at in velit.

    </p>

    <section class="container">

        <form id="my-form">

            <h1>Add User</h1>

            <div class="msg"></div>

            <div>

                <label for="name">Name:</label>

                <input type="text" id="name" name="name">

            </div>

            <div>

                <label for="email">Email:</label>

                <input type="text" id="email" name="email">

            </div>

            <input class="btn" type="submit" value="Submit">

        </form>

        <ul id="users"></ul>

    </section>

    <script src="main.js"></script>

</body>

</html>

On crée un fichier main.js

const btn = document.querySelector(".btn");

🡺 on récupère le bouton submit

btn.addEventListener('click', e => {

  e.preventDefault();

  console.log(e);

});

🡺 addEventListener : on associe à cet élément un écouteur qui admet un couple de paramètres : un événement et un traitement.

click : l’événement

e => {} : traitement

**NB :** l’événement peut être l’un des éléments de la liste se trouvant dans ce lien

<https://developer.mozilla.org/en-US/docs/Web/Events#event_listing>

**Activité de démonstration**

🡺 travailler sur le traitement suite au déclenchement de l’événement 🡺 empêcher la soumission du formulaire

e.preventDefault();

🡺changer la couleur de l’arrière plan du formulaire (méthode de style en ligne)

document.getElementById('my-form').style.background = '#ccc';

🡺changer la couleur de l’arrière plan de la page (méthode d’ajout de classe)

document.querySelector('body').classList.add('bg-dark');

🡺changer le contenu html d’un élément

document.querySelector("h1").innerHTML = "<h1>H1 changé </h1>";

**[[commenter le bloc precedent pour passer au suivant]]**

**Application réelle**

*// je recupere mon formulaire*

const myForm = document.querySelector("#my-form");

myForm.addEventListener("submit", onSubmit);

On associe à l’élément myForm la fonction onSubmit (qu’on va créer juste après) suite à l’événement submit 🡺 implementer la fonction onSubmit

function onSubmit(e) {

}

🡺 empecher la soumission du formulaire

e.preventDefault();

🡺 controler les champs du formulaire : tester si les champs son vides cad qu’il y a des erreurs

if (nameInput.value === "" || emailInput.value === "") {

  }

🡺 traiter le cas où il y a des erreurs

msg.innerHTML = " Merci de compléter tous les champs";

msg.classList.add("error");

    setTimeout(() => {

      msg.innerHTML = "";

      msg.classList.remove("error");

    }, 2000);

msg.classList.add("error"); : on ajoute la classe error pour afficher le message en orange

msg.innerHTML : la chaine est vide au début, on lui attribue le message d’erreur

setTimeout : traitement exécuté apres x temps (dans cet exemple 2000ms = 2 secondes)

🡺traiter le cas où il y en a pas

else {

    const li = document.createElement("li");

    li.appendChild(

      document.createTextNode(`${nameInput.value}: ${emailInput.value}`)

    );

    li.innerHTML = `<strong>${nameInput.value}</strong> ${emailInput.value}`;

    userList.appendChild(li);

    nameInput.value = "";

    emailInput.value = "";

  }

const li = document.createElement("li"); : création d’une var li qui contient un noed <li>

li.appendChild(

      document.createTextNode(`${nameInput.value}: ${emailInput.value}`)

    ); : creation d’un contenu textuel et le mettre à l’intérieur de li

li.innerHTML = `<strong>${nameInput.value}</strong> ${emailInput.value}`; : autre possibilité de mettre du contenu dans le li, au lieu de ne mettre que du texte, on peut mettre du contenu html

userList.appendChild(li); : on insère la li dans le ul

nameInput.value = ""; et emailInput.value = ""; : on efface les valeurs saisies en mettant à la chaine vide pour permettre une nouvelle saisie.