**JavaScript**

Table des matières

[Liaison à une page HTML 7](#_Toc124079641)

[Sélection d’un élément HTML 7](#_Toc124079642)

[Sélection par identifiant : *document.getElementById()* 7](#_Toc124079643)

[Sélection par classe : *document.getElementByClassName()* 7](#_Toc124079644)

[Possibilités d’affichage de données 7](#_Toc124079645)

[Affichage dans la page *.write()* 7](#_Toc124079646)

[Affichage dans un élément HTML *.innerHTML* 7](#_Toc124079647)

[Affichage dans une fenêtre d’alerte : *window.alert()* 7](#_Toc124079648)

[Affichage dans la console de debug : *console.log()* 8](#_Toc124079649)

[Opérateurs 8](#_Toc124079650)

[Types d’opérateurs *(+, %, ==, >=* …*)* 8](#_Toc124079651)

[Et/ou *(&&, ||)* 8](#_Toc124079652)

[Incrémentation *(++)* et décrémentation*(--)* 8](#_Toc124079653)

[Opération réalisable ou non : *isNaN()* 8](#_Toc124079654)

[Opérateur conditionnel *(?)* 8](#_Toc124079655)

[Variables 9](#_Toc124079656)

[Différence entre déclarer et assigner une variable 9](#_Toc124079657)

[Les différentes variables déclarables : *var, let, const* 9](#_Toc124079658)

[Différence entre var et let 9](#_Toc124079659)

[Principe de types dynamiques 9](#_Toc124079660)

[Principe de la variable de tableau constant (const) 10](#_Toc124079661)

[Types de données (Number, String, Object…) 10](#_Toc124079662)

[Déterminer le type de données : *typeof* 10](#_Toc124079663)

[Différence entre *null* et *undefined* 10](#_Toc124079664)

[Conversions de types 11](#_Toc124079665)

[Convertir en chaine *.toString(), .join()* 11](#_Toc124079666)

[Convertir en nombre : *parseInt(), parseFloat()* 11](#_Toc124079667)

[Chaines de caractères 11](#_Toc124079668)

[Déterminer la longueur d’une chaîne de caractères *.length* 11](#_Toc124079669)

[Ajouter des " dans une chaîne de caractères 11](#_Toc124079670)

[Mélanger chaine de caractères et variables 11](#_Toc124079671)

[Extraction d’une partie de la chaîne de caractères .*slice(), .substring(), .substr()* 11](#_Toc124079672)

[Extraction de caractères dans la chaine *.charAt(), .charCodeAt()* 12](#_Toc124079673)

[Conversion de chaine en tableau *.split()* 12](#_Toc124079674)

[Remplacement d’un morceau de contenu par un autre contenu .*replace()* 12](#_Toc124079675)

[Conversion majuscules/minuscules .*toUpperCase(), .toLowerCase()* 12](#_Toc124079676)

[Assemblage de plusieurs chaînes de caractères .*concat()* 12](#_Toc124079677)

[Ajout de caractères à une extrémité de la chaine *.padStart(), .padEnd()* 12](#_Toc124079678)

[Suppression des espaces sur les côtés d’une chaine de caractères .*trim()* 12](#_Toc124079679)

[Recherche dans une chaine de caractères .indexOf() , .lastIndexOf(), .match(), .includes(), .startsWith(), .endsWith() 13](#_Toc124079680)

[Nombres 13](#_Toc124079681)

[Trouver la plus petite/plus grande valeur d’un liste de nombres : *Math.min(), Math.max()* 13](#_Toc124079682)

[Déterminer le signe (positif, négatif, nul) d’un nombre : *Math.signe()* 13](#_Toc124079683)

[Choisir la longueur du nombre *.toPrecision()* 13](#_Toc124079684)

[Choisir le nombre de chiffres après la virgule *.toFixed()* 13](#_Toc124079685)

[Afficher sous la forme exponentielle *.toExponential()* 13](#_Toc124079686)

[Arrondir pour obtenir un nombre entier : *Math.round(), Math.ceil(), Math.floor(), Math.trunc()* 14](#_Toc124079687)

[Puissance, racine carrée, valeur absolue, sinus, cosinus : *Math.pow() , Math.sqrt(), Math.abs(), Math.sin(), Math.cos()* 14](#_Toc124079688)

[Renvoyer un nombre aléatoire entre 0 et 1 : *Math.random()* 14](#_Toc124079689)

[Booléens 14](#_Toc124079690)

[Déclarer un booléen : *new Boolean()* 14](#_Toc124079691)

[Dates 15](#_Toc124079692)

[Initialiser une date : *new Date()* 15](#_Toc124079693)

[Afficher une date dans un autre format *.toDateString(), .toUTCString(), .toISOString()* 15](#_Toc124079694)

[Obtenir une donnée en particulier dans une date *.getFullYear(), .getMonth(), .getHours(), .getSeconds(), .getTime(), .getDay()* … 15](#_Toc124079695)

[Modifier une partie de la date *.setDate(), .setFullYear(), .setHours(), .setMinutes(), .setSeconds(), .setTime()* … 15](#_Toc124079696)

[Tableaux 16](#_Toc124079697)

[Initialiser un tableau 16](#_Toc124079698)

[Obtenir le nombre d’éléments du tableau *.length* 16](#_Toc124079699)

[Accéder et modifier un élément du tableau : *tab[]* 16](#_Toc124079700)

[Afficher le tableau complet 16](#_Toc124079701)

[Ajouter un élément au début du tableau + décalage des index *.unshift()* 16](#_Toc124079702)

[Ajouter un élément à la fin du tableau : *push()* 16](#_Toc124079703)

[Supprimer le premier élément du tableau + décalage des index *.shift()* 16](#_Toc124079704)

[Supprimer le dernier élément du tableau : *pop()* 16](#_Toc124079705)

[Ajouter des éléments n’importe où dans le tableau *.splice()* 17](#_Toc124079706)

[Supprimer des éléments n’importe où dans le tableau *.splice()* 17](#_Toc124079707)

[Fusion de 2 tableaux *.concat()* 17](#_Toc124079708)

[Extraction d’un morceau du tableau *.slice()* 17](#_Toc124079709)

[Rechercher dans un tableau selon l’indice *.indexOf()* 17](#_Toc124079710)

[Vérifier si une valeur est contenue dans le tableau .includes() 17](#_Toc124079711)

[Trier un tableau *.sort()* 18](#_Toc124079712)

[Trier un tableau d’objets *.sort()* 18](#_Toc124079713)

[Trouver la plus grande/petite valeur du tableau : *Math.max.apply(), Math.min.apply()* 18](#_Toc124079714)

[Inverser l’ordre d’un tableau *.reverse()* 18](#_Toc124079715)

[Obtenir les paires index/valeurs d’un tableau *.entries()* 18](#_Toc124079716)

[Appliquer une fonction de rappel sur chaque élément du tableau *.forEach(), .map()* 19](#_Toc124079717)

[Effectuer un test sur chaque valeur du tableau *.some(), .every(), .find(), .findIndex()* 19](#_Toc124079718)

[Créer un tableau à partir d’un chaîne *.from()* 19](#_Toc124079719)

[Vérifier si une variable est un objet ou un tableau : Array*.isArray()* 19](#_Toc124079720)

[Objets 19](#_Toc124079721)

[Initialiser un objet 19](#_Toc124079722)

[Ajouter une propriété à un objet 20](#_Toc124079723)

[Supprimer une propriété d’un objet 20](#_Toc124079724)

[Utiliser une propriété d’un objet dans une variable 20](#_Toc124079725)

[Parcourir les propriétés d’un objet 20](#_Toc124079726)

[Objets imbriqués 20](#_Toc124079727)

[Tableau d’objets imbriqués 21](#_Toc124079728)

[Fonction d’objet 21](#_Toc124079729)

[Ajouter une nouvelle fonction à un objet 21](#_Toc124079730)

[Affichage des propriétés d’un objet : *JSON.stringify()* 21](#_Toc124079731)

[Getters et setters d’objets 22](#_Toc124079732)

[Constructeur d’objets 22](#_Toc124079733)

[Prototype d’un objet *.prototype* 22](#_Toc124079734)

[Ensembles 22](#_Toc124079735)

[Initialiser un ensemble : *new Set()* 22](#_Toc124079736)

[Ajouter dans un ensemble *.add()* 23](#_Toc124079737)

[Accéder aux valeurs des éléments d’un ensemble *.values()* 23](#_Toc124079738)

[Accéder aux valeurs des éléments d’un ensemble une par une *.forEach()* 23](#_Toc124079739)

[Cartes 23](#_Toc124079740)

[Initialiser une carte : *new Map()* 23](#_Toc124079741)

[Afficher les paires clé/valeur de la carte *.entries()* 23](#_Toc124079742)

[Obtenir une valeur de la carte *.get()* 24](#_Toc124079743)

[Modifier une valeur dans une carte *.set()* 24](#_Toc124079744)

[Récupérer uniquement les clés / les valeurs de la carte *.keys(), .values()* 24](#_Toc124079745)

[Obtenir la taille de la carte *.size* 24](#_Toc124079746)

[Supprimer un élément de la carte *.delete()* 24](#_Toc124079747)

[Nettoyer une carte de tous ses éléments *.clear()* 24](#_Toc124079748)

[Vérifier l’existence d’une clé particulière dans la carte *.has()* 24](#_Toc124079749)

[Utiliser des objets comme clés 24](#_Toc124079750)

[Fonctions 25](#_Toc124079751)

[Déclarer une fonction : *function* 25](#_Toc124079752)

[Appeler une fonction 25](#_Toc124079753)

[Variables déclarées dans une fonction 25](#_Toc124079754)

[Paramètre par défaut 25](#_Toc124079755)

[Fonctions asynchrones 25](#_Toc124079756)

[Délai d’attente d’une fonction : *setTimeout()* 25](#_Toc124079757)

[Intervalle de répétition d’une fonction : *setInterval()* 26](#_Toc124079758)

[Fonction de rappel 26](#_Toc124079759)

[Promesse : *Promise()* 26](#_Toc124079760)

[Raccourcis asynchrones : *async, await* 27](#_Toc124079761)

[Classes 27](#_Toc124079762)

[Déclarer une classe et son constructeur : *class, constructor()* 27](#_Toc124079763)

[Créer des objets appartenant à la classe : *new* 27](#_Toc124079764)

[Déclarer des fonctions de classes 27](#_Toc124079765)

[Héritage de classe : *extends* 28](#_Toc124079766)

[Getters et Setters de classe *: get, set* 28](#_Toc124079767)

[Boucles répétitives (for/while) 28](#_Toc124079768)

[Structure d’une boucle avec *for()* 28](#_Toc124079769)

[Structure d’une boucle itérative avec *for(of)* 29](#_Toc124079770)

[Structure d’une boucle avec *while()* 29](#_Toc124079771)

[Structure d’une boucle avec *do{} while()* 29](#_Toc124079772)

[Faire une pause dans une boucle : *break* 29](#_Toc124079773)

[Ignorer une répétition dans la boucle : *continue* 29](#_Toc124079774)

[Boucles conditionnelles (if) 30](#_Toc124079775)

[Structure d’une boucle conditionnelle : *if(), else if(), else()* 30](#_Toc124079776)

[Boucles commutatives (switch) 30](#_Toc124079777)

[Structure d’une boucle commutative : *switch(), case, break, default* 30](#_Toc124079778)

[Gestion des erreurs possibles 30](#_Toc124079779)

[Structure de gestion d’erreurs : *try, catch* 30](#_Toc124079780)

[Générer une erreur personnalisée : *throw* 31](#_Toc124079781)

[Evènements 31](#_Toc124079782)

[Quelques événements possibles *: onchange, onclick, onkeydown…* 31](#_Toc124079783)

[Impression de la page 31](#_Toc124079784)

[Préparer une impression de la fenêtre : *window.print()* 31](#_Toc124079785)

[DOM HTML (modifier les éléments de la page Web) 31](#_Toc124079786)

[Rechercher un élément HTML selon son identifiant *.getElementById()* 32](#_Toc124079787)

[Rechercher des éléments par nom de balise *.getElementsByTagName()* 32](#_Toc124079788)

[Rechercher des éléments par nom de classe *.getElementsByClassName()* 32](#_Toc124079789)

[Rechercher des éléments par ensemble de balise et de classe *.querySelectorAll()* 32](#_Toc124079790)

[Modifier le contenu d’un élément *.innerHTML* 32](#_Toc124079791)

[Modifier la valeur d’un attribut HTML , ou de sa propriété CSS 32](#_Toc124079792)

[Evènements d’arrivée ou de sortie de la page : *onload* et *onunload* 32](#_Toc124079793)

[Evénement de saisie : *onchange* 33](#_Toc124079794)

[Evénements liés à la souris : *onmousedown, onmouseup, onclick* 33](#_Toc124079795)

[Attacher un gestionnaire d’évènement à une fonction : *addEventListener()* 33](#_Toc124079796)

[Ordre d’exécution de gestionnaires d’événements imbriqués (par bulles ou par capture) 33](#_Toc124079797)

[Supprimer un gestionnaire d’événement : *removeEventListener()* 33](#_Toc124079798)

[Validation de formulaire avec JavaScript 33](#_Toc124079799)

[BOM HTML (fenêtre du navigateur) 34](#_Toc124079800)

[L’objet fenêtre : *window* 34](#_Toc124079801)

[Pop-ups pour l’utilisateur : *window.alert, window.confirm, window.prompt* 34](#_Toc124079802)

[Récupérer la taille de la fenêtre *.innerWidth, .innerHeight* 34](#_Toc124079803)

[Méthodes de fenêtre *.open(), .close(), .moveTo(), .resizeTo()* 34](#_Toc124079804)

[Propriétés d’emplacement de la fenêtre 34](#_Toc124079805)

[Naviguer dans l’historique du navigateur *.history.back(), .history.forward()* 35](#_Toc124079806)

[Propriétés de l’écran *.width, .heigth, .availWidth, .availHeigth* 35](#_Toc124079807)

[Gestion des cookies 35](#_Toc124079808)

[Vérifier l’activation des cookies : *navigator.cookieEnabled* 35](#_Toc124079809)

[Créer un cookie : *document.cookie* 35](#_Toc124079810)

[Fonction de création de cookie 36](#_Toc124079811)

[Récupérer la liste des cookies 36](#_Toc124079812)

[Le format JSON 36](#_Toc124079813)

[Syntaxe du JSON 36](#_Toc124079814)

[Convertir un objet JavaScript en JSON : *JSON.stringify()* 36](#_Toc124079815)

[Convertir du JSON en objets JavaScript : *JSON.parse()* 37](#_Toc124079816)

[La méthode JSONP 37](#_Toc124079817)

[API Web (Application Programming Interface) 38](#_Toc124079818)

[API de validation de formulaire 38](#_Toc124079819)

[API d’historique 39](#_Toc124079820)

[API de stockage 40](#_Toc124079821)

[API de travail (Web Worker) 40](#_Toc124079822)

[API de récupération (API Fetch) 41](#_Toc124079823)

[API de géolocalisation 41](#_Toc124079824)

[AJAX 42](#_Toc124079825)

[Schéma de fonctionnement 42](#_Toc124079826)

[Envoyer une demande HttpRequest au serveur Web (GET) *.onload, .open(), .send()* 42](#_Toc124079827)

[Envoyer des données au serveur Web (POST) *.setRequestHeader()* 43](#_Toc124079828)

[Gérer plusieurs tâches AJAX dans un même site web 43](#_Toc124079829)

[Récupérer la réponse du serveur *.responseText, .responseXML* 43](#_Toc124079830)

[Récupérer les informations d’en-tête de la réponse du serveur *.getAllResponseHeaders(), .getResponseHeader()* 44](#_Toc124079831)

[Afficher des informations depuis un fichier JSON 44](#_Toc124079832)

[Afficher des informations sous forme de tableau à partir d’un fichier XML 45](#_Toc124079833)

[Exemple d’utilisation : Afficher des recommandations lors d’une saisie avec AJAX-PHP 45](#_Toc124079834)

[Export/Import de code 46](#_Toc124079835)

[Exporter du code : *export* 47](#_Toc124079836)

[Importer du code : *import* 47](#_Toc124079837)

[Mode strict 47](#_Toc124079838)

[Déclarer le mode strict 47](#_Toc124079839)

[Quelques règles du mode strict 47](#_Toc124079840)

# Liaison à une page HTML

<body>

    ...

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

<noscript>Désolé, votre navigateur ne prend pas en charge le Javascript</noscript>

</body>

**Remarque** : il est conseillé de placer les balises de script externe juste avant la balise fermante du <body>, car cela évite de ralentir l’affichage lors de leur traitement

**Remarque**: En cas de non-chargement du script, la balise <noscript> affiche un message sur la page

# Sélection d’un élément HTML

## Sélection par identifiant : *document.getElementById()*

document.getElementById("identifiant");   /\* cherche l'élément qui a l'id "identifiant" \*/

## Sélection par classe : *document.getElementByClassName()*

document.getElementsByClassName("classe");   // cherche les éléments qui ont la classe "classe"

# Possibilités d’affichage de données

## Affichage dans la page *.write()*

document.write("bonjour");

**Remarque**: Cette méthode ne peut être utilisée qu’à des fins de test !!

## Affichage dans un élément HTML *.innerHTML*

document.getElementById("calcul").innerHTML = 5 + 6;    /\* affiche 11 au niveau de l’élément ayant l'id "calcul" \*/

## Une image contenant texte Description générée automatiquementAffichage dans une fenêtre d’alerte : *window.alert()*

window.alert("stop !");

alert("stop !"); // fonctionnement identique

## Affichage dans la console de debug : *console.log()*

console.log(5+2);

# Opérateurs

## Types d’opérateurs *(+, %, ==, >=* …*)*

let x = 10;        // affectation

let a = 10 + 3;    // addition

let b = 10 - 3;    // soustraction

let c = 10 \* 3;    // multiplication

let d = 10 / 3;    // division

let e = 10 % 3;    // reste de la division

let c = 2 \*\* 4;    // puissance (2 puissance 4)

if(f==5){}         // est égal à

if(f!=5){}         // est différent de

if(f>5){}          // est supérieur à

if(f<5){}          // est inférieur à

if(f>=5){}         // est supérieur ou égal à

## Et/ou *(&&, ||)*

if(f==5 && a==2){}         // les 2 conditions doivent être vraies

if(f==5 || a==2){}         // au moins 1 des 2 conditions doit être vraie

## Incrémentation *(++)* et décrémentation*(--)*

var a = 5;

a++;        //a=6

a--;        //a=4

## Opération réalisable ou non : *isNaN()*

var n = 100 / "pomme";     // n va renvoyer NaN, car l'opération n'est pas réalisable

isNaN(n);                  // renvoie true si l'opération n'est pas réalisable

## Opérateur conditionnel *(?)*

var peutVoter = (age < 18) ? "trop jeune":"assez agé";  // peutVoter prendra la valeur "trop jeune" si la condition est vraie

# Variables

## Différence entre déclarer et assigner une variable

var x;            // la variable est déclarée

x = 10;           // la variable est assignée

## Les différentes variables déclarables : *var, let, const*

var x = 2;         /\* variable dont la valeur peut changer \*/

let name = 'John Doe';       /\* variable dont la valeur peut changer \*/

const salutationMatin = "Bonjour !";      /\* variable avec une référence constante \*/

let person = "John Doe", carName = "Volvo", price = 200;     /\* déclaration de plusieurs variables de type let \*/

**Remarque**: Si on déclare une variable sans lui donner de valeur, sa valeur par défaut est *undefined.*

**Remarque** : «var» ne s’utilise plus trop, on préfère utiliser « let » pour déclarer une variable classique

## Différence entre var et let

var jaune = 'yellow';

var rouge = 'red';

if(rouge == 'red'){

    var jaune = 'black';

    let rouge = 'green';

    document.write(jaune);    // affiche black

    document.write(rouge);    // affiche green

}

document.write(jaune);    // affiche black (la variable a été changé définitivement)

document.write(rouge);    // affiche red (la variable a été changé le temps de la boucle if)

**Remarque**: Les variables globales (déclarées en dehors de fonctions) possèdent une durée de vie équivalente à celle de la page : elles ne sont supprimées que lorsque la page est actualisée ou fermée.

## Principe de types dynamiques

En JavaScript, une même variable peut être utilisée pour contenir différents types de données succinctement

let x;           // x est indéfini

x = 5;           // x est considéré comme un nombre

x = "John";      // maintenant x est considéré comme une chaine de caractères

## Principe de la variable de tableau constant (const)

On peut modifier les éléments d’un tableau constant :

const voitures = ["Peugeot", "Volvo", "BMW"];

voitures[0] = "Toyota" ;       // On peut changer la valeur d'un élément du tableau

voitures.push("Audi");         // On peut ajouter un élément dans le tableau

Mais on ne peut pas réaffecter directement le tableau :

const voitures = ["Peugeot", "Volvo", "BMW"];

voitures = ["Toyota", "Volvo", "BMW", "Audi"];  //Impossible

## Types de données (Number, String, Object…)

let length = 16;                               // Nombre

let length = 16.25;                           // Nombre (avec décimales)

let lastName = "Johnson";                       // Chaine de caractères

let x = {firstName:"John", lastName:"Doe"};     // Objet

const voitures = ["Volvo", "Audi", "Peugeot"];    // Tableau

const d = new Date(2018, 11, 24, 10, 33, 30, 0);   // Date (année, mois, jour, heure, minute, secondes, millisecondes)

(x == y);     // booléen (retourne Vrai ou Faux)

## Déterminer le type de données : *typeof*

typeof "John Doe";     // Retourne "string"

typeof "";     // Retourne "string"

typeof 3.14;           // Retourne "number"

typeof function maFonction(){}    // Retourne "function"

var voiture;

typeof voiture;       // Retourne "undefined" car la variable n'a pas de valeurs

**Remarque**: le type de données de *NaN* est un nombre, le type de données d’un tableau est un objet, le type de données d’une date est un objet, le type de données de *null* est un objet.

## Différence entre *null* et *undefined*

var personne ={prénom:"John", nom:"Doe", age:50};

personne = null;          // la valeur est nulle, mais le type de variable est toujours "object"

personne =  undefined;    // la valeur et le type de variable sont indéfinis

# Conversions de types

## Convertir en chaine *.toString(), .join()*

const fruits = ["banane", "pomme", "cerise"]

fruits.toString();      // une chaine qui contient "banane, pomme, cerise" est retournée

fruits.join("\*");       // renvoie aussi une chaine, mais en spécifiant le séparateur : banane\*pomme\*cerise

## Convertir en nombre : *parseInt(), parseFloat()*

var n = "50"             // n est une chaine de caractères

Number(n) // renvoie 50

var i = parseInt(n);     // i est un nombre entier

var f = parseFloat(n);   // f est un nombre flottant

**Remarque** : Si on ne peut pas convertir en nombre, NaN est renvoyé

# Chaines de caractères

## Déterminer la longueur d’une chaîne de caractères *.length*

var c = "bonjour hihihi";

var longueur = c.length; // renvoie 14

## Ajouter des " dans une chaîne de caractères

var c = 'nous sommes appelé "les codeurs" dans le milieu';

## Mélanger chaine de caractères et variables

let tempExt = 36;

let tempInt = 21;

let meteo = "Il fait " + tempExt + " degrés dehors et " + tempInt + " degrés içi";

let meteoAutrement = `Il fait ${tempExt} degrés dehors et ${tempInt} degrés içi`;        // exactement la même chose

**Remarque**: l’utilisation de ${} est très pratique pour éviter la répétition de +, et pour gérer les espaces comme une seule et vraie phrase

## Extraction d’une partie de la chaîne de caractères .*slice(), .substring(), .substr()*

var fruits = "fraise, banane, pomme";

var extract1 = fruits.slice(16, 22);       // extrait la partie allant du 16ème caractère (inclus) au 22ème caractère (exclu)

var extract1bis = fruits.slice(-1, -6);    // cela fonctionne aussi avec des valeurs négatives (on part de la fin de la chaîne)

var extract2 = fruits.substring(16, 22);   // extrait comme slice, mais les valeurs négatives deviennent égales à 0

var extract3 = fruits.substr (16, 5);      // le 1er paramètre est le caractère de départ, le second est la longueur du morceau extrait

## Extraction de caractères dans la chaine *.charAt(), .charCodeAt()*

var texte = "Hello";

var renvoiIndex = texte.charAt(1);      // Renvoie le caractère en 2ème position (e)

var renvoiCode = texte.charCodeAt(3);   // Renvoie le code UTF-16 du caractère en 4ème position

## Conversion de chaine en tableau *.split()*

var caracteres = "a,b,c,d,e,f";

const tableau = caracteres.split(",");     // chaque case du tableau contient un bout de la chaine, selon les séparations qui sont ","

const tableau2 = caracteres.split(" ");    // les séparations de la chaine sont déterminées par les espaces

Remarque : Si on oublie d’indiquer le séparateur, chaque case du tableau contiendra un caractère de la chaine

## Remplacement d’un morceau de contenu par un autre contenu .*replace()*

var phrase = "allez visiter Microsoft Doc !";

var r1 = phrase.replace("Microsoft", "w3Schools");         // permet de remplacer la 1ère occurence de "Microsoft" rencontrée

var r2 = phrase.replace(/MICROSOFT/i, "w3Schools");        // permet d'éviter de devoir respecter la casse

var r3 = phrase.replace(/Microsoft/g, "w3Schools");        // permet de remplacer toutes les correspondances de "Microsoft" d'un coup

## Conversion majuscules/minuscules .*toUpperCase(), .toLowerCase()*

var texte = "Hello World !";

var maj = texte.toUpperCase();       // toute la chaîne de caractères est en majuscules

var min = texte.toLowerCase();       // toute la chaîne de caractères est en minuscules

## Assemblage de plusieurs chaînes de caractères .*concat()*

var texte1 = "Hello";

var texte1 = "World";

var textConcat = texte1.concat(" ", texte2);    // texteConcat = "Hello World"

## Ajout de caractères à une extrémité de la chaine *.padStart(), .padEnd()*

var texte = "Hello";

var ajoutX = texte.padStart(4,"X");    // Ajout au début de la chaine (en sortie : XXXHello)

var ajoutY = texte.padEnd(4,"Y");      // Ajout à la fin de la chaine (en sortie : HelloYYY)

## Suppression des espaces sur les côtés d’une chaine de caractères .*trim()*

var texte = "    Hello    ";

var enlevEspace = texte.trim();

var enlevEspaceDev = texte.trimStart();     //enlève les espaces au début

var enlevEspaceDer = texte.trimEnd();       //enlève les espaces à la fin

## Recherche dans une chaine de caractères .indexOf() , .lastIndexOf(), .match(), .includes(), .startsWith(), .endsWith()

var texte = "la localisation dans la localisation";

texte.indexOf("localisation");                   // renvoie l'index de la première occurence trouvée (ici 3)

texte.indexOf("localisation", 15);               // renvoie l'index de la première occurence trouvée à partir du 16ème caractère

texte.lastIndexOf("localisation");               // renvoie l'index de la dernière occurence trouvée (ici 24)

texte.match(/sation/g);                          // renvoie toutes les correspondances comportant "sation"

texte.match(/sation/gi);                         // rend la recherche insensible à la casse

texte.includes("local")                          // renvoie "true" si l'extrait est compris dans la chaine

texte.includes("local", 5)                       // renvoie "true" si l'extrait est compris dans la chaine à partir du 6ème caractère

texte.startsWith("la")                           // renvoie "true" si la chaine commence par l'extrait

texte.startsWith("localisation", 3)              // renvoie "true" si le 4ème caractère commence comme l'extrait

texte.endsWith("sation")                         // renvoie "true" si la chaine finit par l'extrait

# Nombres

## Trouver la plus petite/plus grande valeur d’un liste de nombres : *Math.min(), Math.max()*

Math.min(0, 150, 30, 20, -8, -200);   // renvoie -200

Math.max(0, 150, 30, 20, -8, -200);   // renvoie 150

## Déterminer le signe (positif, négatif, nul) d’un nombre : *Math.signe()*

Math.sign(5);       // renvoie 1 si le nombre est positif, 0 s'il est nul et -1 s'il est négatif

## Choisir la longueur du nombre *.toPrecision()*

var n = 9.579;

n.toPrecision(2);        // 2 chiffres en tout (9.6)

n.toPrecision(3);        // 3 chiffres en tout (9.58)

## Choisir le nombre de chiffres après la virgule *.toFixed()*

var n = 9.579;

n.toFixed(2);        // 2 chiffres après la virgule (9.57)

n.toFixed(5);        // 2 chiffres après la virgule (9.57900)

## Afficher sous la forme exponentielle *.toExponential()*

var n = 9.579;

n.toExponential(2);     // 2 chiffres après la virgule (9.58e+0)

n.toExponential(4);     // 4 chiffres après la virgule (9.5790e+0)

**Remarque**: Si on ne précise pas le nombre de chiffres après la virgule, le nombre ne sera pas arrondi

## Arrondir pour obtenir un nombre entier : *Math.round(), Math.ceil(), Math.floor(), Math.trunc()*

Math.round(4.8);    // arrondi à l'entier le plus proche (renvoie 5)

Math.round(4.5);    // renvoie 5

Math.round(4.3);    // renvoie 4

Math.ceil(4.9);     // arrondi à l'entier supérieur (renvoie 5)

Math.ceil(4.1);     // renvoie 5

Math.ceil(-4.9);    // renvoie -4

Math.floor(4.9);    // arrondi à l'entier inférieur (renvoie 4)

Math.floor(4.1);    // renvoie 4

Math.floor(-4.9);   // renvoie -5

Math.trunc(4.4);    // renvoie la partie entière du nombre (renvoie 4)

Math.trunc(-4.7);   // renvoie -4

## Puissance, racine carrée, valeur absolue, sinus, cosinus : *Math.pow() , Math.sqrt(), Math.abs(), Math.sin(), Math.cos()*

Math.pow(4,2);                    // renvoie 4 à la puissance 2 (16)

Math.sqrt(64);                    // renvoie la racine carrée de 64 (8)

Math.abs(-7.5);                   // renvoie la valeur absolue de -7.5 (7.5)

Math.sin(90 \* Math.PI / 180);     // renvoie le sinus de 90 degrés (1)

Math.cos(0 \* Math.PI / 180);      // revoie le cosinus de 0 degrés (1)

## Renvoyer un nombre aléatoire entre 0 et 1 : *Math.random()*

Math.random();

Math.floor(Math.random()\*10);       // retourne un entier aléatoire entre 0 et 9

Math.floor(Math.random()\*101);      // retourne un entier aléatoire entre 0 et 100

Math.floor(Math.random()\*10)+1;     // retourne un entier aléatoire entre 1 et 10

Math.floor(Math.random()\*100)+1;    // retourne un entier aléatoire entre 1 et 100

function getEntierAlea(min, max) {

    return Math.floor(Math.random() \* (max - min + 1) ) + min;   // retrourne un nombre aléatoire entre min et max inclus

}

## Effectuer un calcul à partir d’une chaine de caractères : eval()

let calcul = "(4+2)/3";           // "calcul" est une chaine de caractères

let resultat = eval(calcul);      // "resultat" contient 2, soit le résultat du calcul

# Booléens

## Déclarer un booléen : *new Boolean()*

var x = false;

# Dates

## Initialiser une date : *new Date()*

const d = new Date(2018, 11, 24, 10, 33, 30, 0);     // (année, mois, jour, heures, minutes, secondes, millisecondes)

const d = new Date(2018, 11, 24, 10, 33); // (année, mois, jour, heures, minutes)

const d = new Date(2018, 11); // (année, mois)

const d = new Date("October 13, 2014 11:13:00"); // fonctionne également

**Remarque :** en JavaScript, les mois vont de 0 (pour janvier) à 11 (pour décembre). Si la valeur du mois est supérieure à 11, l’écart est reporté sur l’année d’après

**Remarque** : il est possible de comparer 2 dates, dans le contexte d’un boucle *if* par exemple

## Afficher une date dans un autre format *.toDateString(), .toUTCString(), .toISOString()*

const date = new Date("September 9, 2022 16:34:06");

date.toDateString();            // "Fri, Sep 09 2022"

date.toUTCString();             // "Fri, 09 Sep 2022 16:34:06 GMT"  --> format UTC

date.toISOString();             // "2022-09-09T16:34:06.226Z"  --> format ISO

## Obtenir une donnée en particulier dans une date *.getFullYear(), .getMonth(), .getHours(), .getSeconds(), .getTime(), .getDay()* …

const date = new Date("September 9, 2022 16:34:06");

date.getFullYear();       // renvoie l'année d'une date sous la forme d'un nombre à 4 chiffres

date.getMonth();          // renvoie le mois d'une date sous la forme d'un nombre (de 0 à 11)

date.getDate();           // renvoie le jour d'une date sous la forme d'un nombre (de 1 à 31)

date.getHours();          // renvoie l'heure d'une date sous la forme d'un nombre (de 0 à 23)

date.getMinutes();        // renvoie la minute d'une date sous la forme d'un nombre (de 0 à 59)

date.getSeconds();        // renvoie la seconde d'une date sous la forme d'un nombre (de 0 à 59)

date.getMilliseconds();   // renvoie les millisecondes d'une date sous la forme d'un nombre (de 0 à 999)

date.getDay();            // renvoie le jour de la semaine d'une date sous la forme d'un nombre (de 0 --> dimanche à 6 --> samedi)

date.getTime();           // renvoie le nombre de millisecondes écoulées depuis le 1er janvier 1970

## Modifier une partie de la date *.setDate(), .setFullYear(), .setHours(), .setMinutes(), .setSeconds(), .setTime()* …

const date = new Date("September 9, 2022 16:34:06");

date.setFullYear(2020);       // la nouvelle année de la variable date est 2020

date.setFullYear(2020, 11, 3);       // la nouvelle année est 2020, le nouveau mois est décembre et le nouveau jour est le 3

date.setMonth(11);          // le nouveau mois est décembre

date.setDate(15);           // la nouvelle date est le 15 du mois

date.setHours(22);

date.setMinutes(30);

date.setSeconds(30);

date.setDay(5);

# Tableaux

La différence entre tableaux et objets est que les tableaux utilisent des index numérotés, alors que les objets utilisent des index nommés.

## Initialiser un tableau

const tab = [ const tab = new Array("HTML","CSS","JS");

    "HTML", // Même chose

    "CSS",

    "JS"

];           // Le tableau contient 3 éléments

**Remarque**: Les éléments contenus dans le tableau peuvent être d’autres objets (ex : fonctions, variables, autre tableau…).

**Remarque**: Il faut toujours déclarer un objet avec const

## Obtenir le nombre d’éléments du tableau *.length*

var taille = tab.length; // retourne le nombre d’éléments dans le tableau

## Accéder et modifier un élément du tableau : *tab[]*

var langage = tab[0];   // la variable contient le premier élément du tableau ("HTML")

var langage = tab[2];   // la variable contient le troisieme élément du tableau ("JS")

var langage = tab[tab.length – 1];   // la variable contient le dernier élément du tableau ("JS")

tab [0] = "Python";     // le premier élément du tableau a changé pour "Python"

## Afficher le tableau complet

document.getElementById("tableau").innerHTML = tab;    // renvoie le tableau dans son intégralité

## Ajouter un élément au début du tableau + décalage des index *.unshift()*

const fruits = ["banane", "pomme", "cerise"]

fruits.unshift("citron");     // "citron" est ajouté au début du tableau, tous les autres éléments sont décalés vers l'index supérieur

## Ajouter un élément à la fin du tableau : *push()*

tab.push("C#");     // l'élément est ajouté à la fin du tableau

## Supprimer le premier élément du tableau + décalage des index *.shift()*

const fruits = ["banane", "pomme", "cerise"]

fruits.shift();     // "banane" est supprimé, tous les autres éléments sont décalés vers l'index inférieur

## Supprimer le dernier élément du tableau : *pop()*

const fruits = ["banane", "pomme", "cerise"]

fruits.pop();                 // le dernier élement est supprimé ("banane", "pomme")

var enlevé = fruits.pop();    // la variable contient la valeur qui a été enlevée du tableau

## Ajouter des éléments n’importe où dans le tableau *.splice()*

const fruits = ["banane", "pomme", "fraise"];

fruits.splice(1, 0, "kiwi", "melon");            //1🡪index de départ des nouveaux éléments , 0🡪 nombre d'éléments à supprimer

// en sortie : "banane", "kiwi", "melon", "pomme", "fraise"

## Supprimer des éléments n’importe où dans le tableau *.splice()*

const fruits = ["banane", "pomme", "fraise", "kiwi"];

fruits.splice(2, 1);                             // 2--> index de départ des éléments à supprimer,   1--> nombre d'éléments à supprimer

                                                 // en sortie : "banane", "pomme", "kiwi"

## Fusion de 2 tableaux *.concat()*

const fruits = ["banane", "pomme"];

const legumes = ["carotte", "oignon"];

const epices = ["curry", "persil"];

const ensemble = fruits.concat(legumes);           //"banane", "pomme", "carotte", "oignon"

const ensemble = fruits.concat(legumes, epices);       //"banane", "pomme", "carotte", "oignon", "curry", "persil"

const ensemble = fruits.concat("ananas", "cerise"); //"banane", "pomme", "ananas", "cerise"

## Extraction d’un morceau du tableau *.slice()*

const fruits = ["banane", "pomme", "fraise", "kiwi"];

const nouveauTab = fruits.slice(2)          // 2--> index du premier élément du nouveau tableau (nouveau tableau: "fraise", "kiwi")

const nouveauTab = fruits.slice(1,3)

// 3--> limite non comprise du nouveau tableau (nouveau tableau: "pomme", "fraise")

## Rechercher dans un tableau selon l’indice *.indexOf()*

const fruits=["pomme", "cerise", "prune" , "cerise"];

var position = fruits.indexOf("cerise") + 1;       // renvoie la position de la première occurrence de cerise : position 2

var positionDernier = fruits.lastIndexOf("cerise") + 1 ;      // renvoie la position de la dernière occurrence de cerise : position 4

**Remarque** : indexOf() et lastIndexOf() renvoient -1 si l’élément n’est pas trouvé

## Vérifier si une valeur est contenue dans le tableau .includes()

const fruits = ["fraise", "pomme", "mangue", "ananas"];

fruits.includes("mangue");       // renvoie true si la valeur est dans le tableau, false sinon

## Trier un tableau *.sort()*

const fruits = ["banane", "pomme", "fraise", "kiwi"];

fruits.sort();                                     // tri de chaines par ordre alphabétique

const nombres = [25, 500, 6, 478];

nombres.sort(function(a, b){return a - b});        //tri de nombres par ordre croissant

nombres.sort(function(a, b){return b - a});        //tri de nombres par ordre décroissant

## Trier un tableau d’objets *.sort()*

const voitures = [

    {marque:"Volvo", annee:2016},

    {marque:"Saab", annee:2001},

    {marque:"BMW", annee:2010}

];

voitures.sort(function(a, b){return a.annee - b.annee});

## Trouver la plus grande/petite valeur du tableau : *Math.max.apply(), Math.min.apply()*

const nombres = [25, 500, 6, 478];

Math.max.apply(nombres);         // renvoie la plus grande valeur (500)

Math.min.apply(nombres);         // renvoie la plus petite valeur (6)

## Inverser l’ordre d’un tableau *.reverse()*

const fruits = ["banane", "pomme", "fraise", "kiwi"];

fruits.reverse();           // inverse tout l'ordre du tableau (renvoie kiwi, fraise, pomme, banane)

## Obtenir les paires index/valeurs d’un tableau *.entries()*

Une image contenant texte

Description générée automatiquementconst fruits = ["fraise", "pomme", "mangue", "ananas"];

const valeurs = fruits.entries();

for(let index of valeurs){

    document.getElementById("paires").innerHTML += index + "</br>";

}

## Appliquer une fonction de rappel sur chaque élément du tableau *.forEach(), .map()*

const nombres = [45, 7, 50, 36];

nombres.forEach(maFonction);       // la fonction est appelée pour chaque élément du tableau

const nombres2 = nombres.map(maFonction);       // en plus d’appliquer sur chaque élément, un nouveau tableau contient les résultats

function maFonction(value, index, array){ // fonction de rappel

    return value \* 2;

}

**Remarque :** Tant que la fonction de rappel utilise le paramètre « value », les paramètres « index » et  « array » peuvent être négligés

## Effectuer un test sur chaque valeur du tableau *.some(), .every(), .find(), .findIndex()*

const nombres = [45, 7, 50, 36];

var tabTest = nombres.some(maFonction); // retourne true si au moins 1 valeur est supérieure à 17

var tabTest2 = nombres.every(maFonction); // retourne true si toutes les valeurs sont supérieures à 17

var tabTest3 = nombres.find(maFonction); // retourne la valeur du premier élément qui réussit le test

var tabTest4 = nombres.findIndex(maFonction); // retourne l’index du premier élément qui réussit le test

function maFonction(value, index, array){ // fonction de rappel

    return value > 17;

}

## Créer un tableau à partir d’un chaîne *.from()*

const nouveau = Array.from("ABCDEF");   // la chaine "ABCDEF" devient le tableau ["A","B","C","D","E","F"]

## Vérifier si une variable est un objet ou un tableau : Array*.isArray()*

**Remarque** : Un tableau est un type d’objet particulier, donc typeOf renverra " object ".

Array.isArray(tab);      // renvoie true si tab est un tableau, false sinon

# Objets

## Initialiser un objet

const person = {

    firstName: "John",        // firstName --> propriété, John --> valeur de la propriété

    lastName: "Doe",

    fullName : function() { return this.firstName + " " + this.lastName; },

    age: 50,

    eyeColor: "blue"

};

**Remarque**: Il faut toujours déclarer un objet avec const

## Ajouter une propriété à un objet

const personne = {} ;               // On initialise un objet vide

personne.nom = "Cauvet";

personne.prenom = "Louis";

personne.age = 20;                 // On affecte un par un des champs à l'objet

## Supprimer une propriété d’un objet

const person = {

    firstName: "John",

    lastName: "Doe",

    age: 50

};

delete person.age; // L’objet ne possède à présent que 2 propriétés

## Utiliser une propriété d’un objet dans une variable

var nbAnnees = person.age;

"la personne est née depuis " + nbAnnees + " années";

## Parcourir les propriétés d’un objet

const person = {

    fname:" John",

    lname:" Doe",

    age: 25

};

for (let x in person) {

    txt += person[x];

}

## Objets imbriqués

personne = {

    nom:"Louis",

    age:20,

    voitures: {

      voiture1:"Mireille Mobile",

      voiture2:"Kangoo rouge"

    }                                 // l'objet voitures est une propriété de l'objet personne

}

personne.voitures.voiture1;           // on peut accéder à un objet imbriqué comme ceci

## Tableau d’objets imbriqués

const person = {

    name: "John",

    age: 30,

    cars: [

      {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},

      {name:"BMW", models:["320", "X3", "X5"]},

      {name:"Fiat", models:["500", "Panda"]}

    ] // la propriété cars est un tableau qui contient plusieurs tableaux

}

## Fonction d’objet

const person = {

    firstName: "John",

    lastName: "Doe",

    id: 5566,

    fullName: function() { // la fonction fullName appartient à l’objet person, et est stockée comme une propriété

      return this.firstName + " " + this.lastName;

    }

};

var identite = person.fullName();         // on peut appeler la fonction de l'objet comme ceci

## Ajouter une nouvelle fonction à un objet

person.identite = function () {

    return this.firstName + " " + this.lastName;

};

## Affichage des propriétés d’un objet : *JSON.stringify()*

Par défaut, afficher un objet en appelant son nom renvoit toujours « [object Object] ».

const person = {name: "John",city: "New York"};

document.getElementById("demo").innerHTML = person.name + person.city; // on peut accéder directement aux éléments individuellement

let myString = JSON.stringify(person);

document.getElementById("demo").innerHTML = myString;  // on peut récupérer tous les éléments dans une chaîne au format JSON



## Getters et setters d’objets

const personne = {

    nom: "Cauvet",

    prenom: "Louis",

    langue: "fr",

    get lang(){return this.langue;},       // getter

    get langMaj(){return this.langue.toUpperCase();}, // l’utilisation des getters permet une manipulation plus facile des propriétés

    set lang(l){this.langue = l;}           // setter

};

personne.lang = "en";     // change la valeur à l'aide du setter

## Constructeur d’objets

function Personne(n, p, a) {

    this.nom = n;

    this.prenom = p;

    this.age = a;

}

const p1 = new Personne("Cauvet","Louis",20);

const p2 = new Personne("Burghardt","Zoe",19);

p1.couleurYeux = "marrons";         // cette propriété sera ajoutée seulement à p1, pas aux autres instances de Personne

## Prototype d’un objet *.prototype*

Le prototype d’un objet est une propriété Javascript qui permet d’ajouter de nouvelles propriétés ou méthodes dans un constructeur

Personne.prototype.nationalité = "Francaise";         // ra=joute une propriété "nationalité" dans le constructeur de Personne

**Remarque** : La comparaison de 2 objets JavaScript est toujours fausse !!!

# Ensembles

Un ensemble est une collection de valeurs uniques, chaque valeur ne peut apparaître qu’une fois dans l’ensemble

## Initialiser un ensemble : *new Set()*

const lettres1 = new Set(["a","b","c"]);

const a = "a";

const b = "b";

const c = "c";

const lettres2 = new Set(); // les 2 méthodes aboutissent au même résultat

lettres2.add(a);

lettres2.add(b);

lettres2.add(c);

## Ajouter dans un ensemble *.add()*

const lettres = new Set(["a","b","c","d"]);

lettres.add("d");

**Remarque** : si on ajoute plusieurs fois le même élément, il sera conservé une seule fois

## Accéder aux valeurs des éléments d’un ensemble *.values()*

const lettres = new Set(["a","b","c"]);

let texte = "";

for (const x of lettres.values()) {

  texte += x + "<br>";

}

## Accéder aux valeurs des éléments d’un ensemble une par une *.forEach()*

const lettres = new Set(["a","b","c"]);

lettres.forEach(function(value) {         // parcourt chaque élément de l'ensemble, pour les afficher un à un dans la console

    console.log(value);

})

**Remarque** : Les ensembles sont des objets (typeof d’un ensemble retourne « object »).

# Cartes

Une carte contient des paires clé-valeur rangées selon l’ordre d’insertion (les clés peuvent être n’importe quel type de données)

## Initialiser une carte : *new Map()*

const fruits = new Map([

    ["pommes", 500],

    ["fraises", 300],

    ["kiwis",250]

]);

## Afficher les paires clé/valeur de la carte *.entries()*

Une image contenant texte

Description générée automatiquementconst fruits = new Map([["pommes", 500],["fraises", 300],["kiwis",250]]);

let texte = "";

for (const x of fruits.entries()) {

  texte += x;

}

## Obtenir une valeur de la carte *.get()*

const fruits = new Map([["pommes", 500],["fraises", 300],["kiwis",250]]);

fruits.get("kiwis");    // retourne 250

## Modifier une valeur dans une carte *.set()*

const fruits = new Map([["pommes", 500],["fraises", 300],["kiwis",250]]);

fruits.set("pommes", 200);

## Récupérer uniquement les clés / les valeurs de la carte *.keys(), .values()*

const fruits = new Map([["pommes", 500],["fraises", 300],["kiwis",250]]);

for (const x of fruits.keys()) {

    console.log(x);                // affiche en console "pommes, fraises, kiwis"

}

for (const x of fruits.values()) {

    console.log(x);                // affiche en console "500, 300, 250"

}

## Obtenir la taille de la carte *.size*

const fruits = new Map([["pommes", 500],["fraises", 300],["kiwis",250]]);

fruits.size;      // renvoie 3

## Supprimer un élément de la carte *.delete()*

const fruits = new Map([["pommes", 500],["fraises", 300],["kiwis",250]]);

fruits.delete("fraises");

## Nettoyer une carte de tous ses éléments *.clear()*

const fruits = new Map([["pommes", 500],["fraises", 300],["kiwis",250]]);

fruits.clear();          // la carte existe toujours, mais est vidée

## Vérifier l’existence d’une clé particulière dans la carte *.has()*

const fruits = new Map([["pommes", 500],["fraises", 300],["kiwis",250]]);

fruits.has("fraises");      // renvoie true si la clé existe, false sinon

## Utiliser des objets comme clés

const pommes = {nom: 'pommes'};

const cerises = {nom: 'cerises'};

const fruits = new Map();

fruits.set(pommes, 500); // l’objet est ajouté comme clé dans la carte

fruits.set(cerises, 200);

**Remarque** : Les ensembles sont des objets (typeof d’un ensemble retourne « object »).

# Fonctions

## Déclarer une fonction : *function*

function maFonction(x,y){      // x et y sont les paramètres de la fonction

    return x \* y;

}

## Appeler une fonction

let result = maFonction(5,2);

document.write("Le résultat du calcul est " + result); // affichage du résultat de la fonction

## Variables déclarées dans une fonction

Les variables déclarées dans une fonction ne sont utilisables qu’au sein de cette fonction

function maFonction() {

    let nomVoiture = "Volvo";    // la variable n'est utilisable que dans la fonction maFonction()

}

## Paramètre par défaut

function maFonction(x, y = 10) {

    return x + y;

}

maFonction(5);          // x vaut 5, et comme y n'est pas indiqué il vaut 10 (sa valeur par défaut)

# Fonctions asynchrones

Une fonction asynchrone est une fonction exécutée en parallèle d’autres fonctions (ex : chargement d’un fichier).

## Délai d’attente d’une fonction : *setTimeout()*

setTimeout(maFonction, 3000);         // attend 3 secondes avant d'exécuter la fonction "maFonction"

function maFonction(){

    console.log("Bonjour !!");

}

## Intervalle de répétition d’une fonction : *setInterval()*

let varIntervalle = setInterval(maFonction, 1000);      // appelle la fonction "maFonction" toutes les secondes

function maFonction(){              // fonction qui affiche l'heure exacte au moment où la fonction est appelée

    let d = new Date();

    document.getElementById("date").innerHTML = d.getHours() +","+ d.getMinutes() +","+ d.getSeconds();

}

<button onclick="clearInterval(varIntervalle)">Arrêter le temps</button>    // permet d'arrêter d'appeler automatiquement la fonction

## Fonction de rappel

Une fonction de rappel est une fonction passé en paramètre d’une autre fonction, pour être exécutée à la suite.

function affichage(param){

    document.getElementById("affich").innerHTML = param;

}

function calculateur(num1, num2, rappel){      // "rappel" est une paramètre correspondant à une fonction de rappel

    let somme = num1 + num2;

    rappel(somme);

}

calculateur(5, 2, affichage);                  // la fonction "affichage" est appelée en rappel de la fonction "calculateur"

**Remarque :** les fonctions de rappel sont difficiles à déboguer, donc on préfère passer par un promesse JavaScript pour faire de la programmation asynchrone.

## Promesse : *Promise()*

Une promesse est un objet qui contient à la fois le code producteur (celui qui peut prendre un certain temps à s’exécuter), et les appels au code consommateur (celui qui attend le résultat du code producteur).

let maPromesse = new Promise(function (reussite,echec){     // les paramètres "reussite" et "echec" sont 2 variables

    let x = 0;

    // code producteur, qui prend un certain temps à s'exécuter

    if (x==0){

        reussite("OK");

    } else {

        echec("Erreur");

    }

});

maPromesse.then(

    function(value) { console.log(reussite);},        /\* code à exécuter en cas de succès \*/

    function(error) { console.log(echec);}            /\* code à exécuter en cas d'échec \*/

);

## Raccourcis asynchrones : *async, await*

async function affichage() {               // le mot-clé "async" oblige la fonction à renvoyer une promesse

    let maPromesse = new Promise(function(resolve, reject) {     // "resolve" en cas de succes, "reject" en cas d'échec

      resolve("C'est une belle journée !");

    });

    document.getElementById("affich").innerHTML = await maPromesse;     // cette fonction attend que le résultat de la promesse lui parvienne avant de l'afficher

}

affichage();

**Remarque :**  «resolve» et «reject» sont deux arguments prédéfinis par JavaScript. Dans cet exemple, on a défini que le succès car la promesse est sûre de s’exécuter correctement.

# Classes

## Déclarer une classe et son constructeur : *class, constructor()*

class voiture{

  constructor(nom, année){

    this.name = nom;

    this.year = année;

  }

}

**Remarque :**  Il faut toujours définir le constructeur d’une classe

## Créer des objets appartenant à la classe : *new*

var voiture1 = new voiture("Ford", 2014);

var voiture2 = new voiture("Audi", 2019);      // 2 objets appartenant à la classe "voiture" ont été crées

## Déclarer des fonctions de classes

class voiture{

  constructor(nom, année){

    this.name = nom;

    this.year = année;

  }

  age(){ // fonction de classe

    let date = new Date();                   // la variable est instanciée avec la date d'aujourd'hui

    return date.getFullYear() - this.year;   // on soustrait à l’année actuelle l'année de la voiture pour connaître son âge

  }

}

## Héritage de classe : *extends*

class Vehicule {

    constructor(proprio){

        this.proprietaire = proprio;

    }

    connaitreProprio(){

        return 'Le propriétaire est ' + this.proprietaire;

    }

}

class Velo extends Voiture {          // la classe Velo hérite des attributs et fonction de la classe Vehicule

    constructor(proprio, roues){

        super(proprio);               // super fait référence à un attribut de la classe parent

        this.nbRoues = roues;

    }

}

## Getters et Setters de classe *: get, set*

class Vehicule {

    constructor(proprio){

        this.proprietaire = proprio;

    }

    get proprio(){ // getter

        return this.proprietaire;

    }

    set proprio(p){ // setter

        this.proprietaire = p;

    }

}

**Remarque** : Les getters et les setters ne peuvent pas avoir le même nom que la propriété (proprietaire != proprio)

# Boucles répétitives (for/while)

## Structure d’une boucle avec *for()*

const voitures = ["Audi", "Volvo", "Kia"]

for (var i=0; i<voitures.length; i++){

    var texte = voitures[i] + "</br>";

    document.write(texte);

}

## Structure d’une boucle itérative avec *for(of)*

const voitures = ["Audi", "Volvo", "Kia"]

for (var i of voitures){         // l'instruction va être répétée le nombre d'éléments contenus dans le tableau

    var texte = voitures[i] + "</br>";

    document.write(texte);

}

## Structure d’une boucle avec *while()*

while (i < 10) {

    texte += "Le nombre est " + i;

    i++;

}

document.write(texte);

## Structure d’une boucle avec *do{} while()*

do{

    texte += "Le nombre est " + i;

    i++;

} while(i<10)        // l'éxécution est répétée tant que la condition n'est pas remplie

## Faire une pause dans une boucle : *break*

for (var i=0; i<10; i++){

    if(i==3) {break;}            // la boucle s'arrête si la condition est remplie

    texte = "le nombre est" + i + "</br>";

}

## Ignorer une répétition dans la boucle : *continue*

for (var i=0; i<10; i++){

    if(i==3) {continue;}            // la boucle ignore le passage si la condition est remplie

    texte = "le nombre est" + i + "</br>";

}

# Boucles conditionnelles (if)

## Structure d’une boucle conditionnelle : *if(), else if(), else()*

if (heure < 7){

    message = "bonjour !";

} else if (heure < 18){

    message = "bonsoir !";

} else{

    message = "bonne nuit !";

}

# Boucles commutatives (switch)

## Structure d’une boucle commutative : *switch(), case, break, default*

var n = 2;

switch(n){

    case 1:

        document.write("Le numéro est 1");

        break;

    case 2:

        document.write("Le numéro est 2");

        break;

    default:

        document.write("Le numero n'est pas défini");

}

**Remarque** : la valeur que l’on teste (ici n) et les différents cas doivent être du même type pour que la boucle fonctionne (on ne pourra pas comparer n= "1" et case : 1)

# Gestion des erreurs possibles

## Structure de gestion d’erreurs : *try, catch*

try{

  addlert("Bienvenue !");       // il y a une erreur : addlert au lieu de alert

}

catch(erreur){

  console.error(erreur.message);       // le message d'erreur est alors affiché dans la console

}

finally{

  console.log("le code a bien tourné")     // le contenu sera exécuté à la fin qu’il y ai exception avant ou non

}

**Remarque**: S’il n’y a pas d’erreur dans la boucle *try ,* le code contenu est réalisé normalement

## Générer une erreur personnalisée : *throw*

var chiffre = 6;

if(chiffre!=4){

  throw "La variable ne contient pas le bon numéro";

}

# Evènements

Un évènement peut correspondre à plusieurs actions différentes :

* Une page web qui a fini de se charger
* Un bouton HTML cliqué
* Un champ de saisie HTML modifié…

<button onclick="displayDate()">Quelle heure est-il ?</button>

L’attribut *onclick* du bouton indique quoi faire lorsque l’événement est réalisé (ici appeler la fonction *displayDate()*)

## Quelques événements possibles *: onchange, onclick, onkeydown…*

<button onchange="displayDate()"></button>        <!-- l'élément HTML a changé -->

<button onclick="displayDate()"></button>         <!-- l'élément HTML a été cliqué -->

<button onmouseover="displayDate()"></button>     <!-- l'élément HTML a été survolé par la souris -->

<button onkeydown="displayDate()"></button>       <!-- une touche du clavier a été pressée-->

<button onload="displayDate()"></button>          <!-- le navigateur a fini de charger la page -->

# Impression de la page

## Préparer une impression de la fenêtre : *window.print()*

<body>

    <button onclick="window.print()">Imprimer la page</button>

</body>

# DOM HTML (modifier les éléments de la page Web)

Le DOM HTML est une norme W3C qui correspond à une interface indépendante du navigateur, qui permet aux programmes et aux scripts de mettre à jour dynamiquement le contenu, la structure ou le style d’une page à l’aide de JavaScript. Dans le DOM, tous les éléments HTML sont considérés comme des objets. Une **propriété** est une valeur que l’on peut obtenir ou définir ( pour modifier le contenu d’un élément HTML), une **méthode** est une action que l’on effectue (ajouter un supprimer un élément HTML).

## Rechercher un élément HTML selon son identifiant *.getElementById()*

const element = document.getElementById("calcul");  // récupère l’élément possédant l’identifiant "calcul"

const textElement = document.getElementById("calcul").textContent; // récupère le texte de l’élément possédant l’identifiant "calcul"

**Remarque :** «document» est un objet représentant la page web.

**Remarque :** Si l’élément n’est pas trouvé, «element» contiendra la valeur *Null*

## Rechercher des éléments par nom de balise *.getElementsByTagName()*

const elements = document.getElementsByTagName("p"); // récupère tous les éléments contenus dans des balises <p>

const elements2 = document.getElementsByTagName("p")[0];    // récupère uniquement le premier élément correspondant rencontré

let long = elements2.length;

elements2[long-1];                                         // récupère uniquement le dernier élément correspondant rencontré

const x = document.getElementById("main");       // "x" contient tous les éléments contenus dans l'élément qui a l'id "main"

const elements3 = x.getElementsByTagName("p");   // "elements3" contient tous les éléments contenus dans les balises <p> de "x"

**Remarque** : les éléments récupérés ainsi sont stockés dans une HTMLCollection, et non dans un tableau (on peut utiliser seulement la propriété *.length*).

## Rechercher des éléments par nom de classe *.getElementsByClassName()*

const elements = document.getElementsByClassName("jaune");      // on récupère les éléments possédant la classe "jaune"

## Rechercher des éléments par ensemble de balise et de classe *.querySelectorAll()*

const elements = document.querySelectorAll("p.jaune");     // récupère tous les éléments de classe "jaune" compris dans les balises <p>

## Modifier le contenu d’un élément *.innerHTML*

document.getElementById("calcul").innerHTML = 17;     // modifie le contenu de l’élément possédant l’id "calcul"

document.getElementById("heureActuelle").innerHTML = "Date : " + Date();     // affichage dynamique de l'heure

<h2 onclick="this.innerHTML='Aie ça fait mal !'">Cliquer sur le texte</h2>    // le contenu du texte change lorsque l'on clique dessus

## Modifier la valeur d’un attribut HTML , ou de sa propriété CSS

document.getElementById("monImage").src = "landscape.jpg";     // change la valeur de "src" pour changer la photo

document.getElementById("bleu").style.color= "red";       // change la couleur du texte en rouge

document.getElementById("espion").style.display = "none";      // rend l'élément non considéré par la page

## Evènements d’arrivée ou de sortie de la page : *onload* et *onunload*

<body onload="checkCookies()"></body>    // onload se déclanche dès que la page est chargée

Remarque : *onload* peut être utilisé pour vérifier le type et la version du navigateur, pour adapter la page à ces critères.

## Evénement de saisie : *onchange*

<input type="text" id="fname" onchange="upperCase()"></input>    // upperCase() transforme la saisie du champ en majuscules

## Evénements liés à la souris : *onmousedown, onmouseup, onclick*

<div onmouseover="changerTexte()" onmouseout="mremettreTexteOrigine()" onmouseclick="chargerPage()"></div>

## Attacher un gestionnaire d’évènement à une fonction : *addEventListener()*

document.getElementById("boutonDate").addEventListener("click", afficheDate);   // appelle la fonction lorsque l'on clique sur le bouton

document.getElementById("boutonDate").addEventListener("mouseover", afficheDate);   // appelle la fonction lorsqu'on survole le bouton

document.getElementById("boutonDate").addEventListener("mouseout", afficheDate);   // appelle la fonction lorsqu'on s'enlève du bouton

function afficheDate() {

  document.getElementById("date").innerHTML = Date();

}

**Remarque** : On peut attacher plusieurs gestionnaires d’évènements à un même élément.

## Ordre d’exécution de gestionnaires d’événements imbriqués (par bulles ou par capture)

On suppose que l’on a un élément <div> qui contient un élément <p>, et que tous les 2 sont liés à un événement par un «*click*». Dans le cas d’une propagation par bulles, l’événement de l’élément interne (ici <p>) est effectué en premier, et l’autre événement est effectué ensuite, alors que c’est l’inverse lors d’une propagation par capture.

document.getElementById("monP").addEventListener("click", maFonction);             // propagation par bulles (par défaut)

document.getElementById("monDiv").addEventListener("click", maFonction);           // propagation par bulles (par défaut)

document.getElementById("monP").addEventListener("click", maFonction, true);       // propagation par capture

document.getElementById("monDiv").addEventListener("click", maFonction, true);     // propagation par capture

## Supprimer un gestionnaire d’événement : *removeEventListener()*

document.getElementById("boutonDate").removeEventListener("mousemove", affichePositionSouris);

// Si le bouton est cliqué, la position de la souris arrête d'être mise-a-jour quand on la déplace

## Validation de formulaire avec JavaScript

<form name="myForm" action="/action\_page.php" onsubmit="return validerForm()" method="post">

    <input type="text" name="fname" required>

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

</form>

function validerForm(){

    let valeur = document.forms["monFormulaire"]["fname"].value    // "fname" désigne un champ de formulaire

    if(valeur == ""){

        alert("on ne peut pas laisser de champs vide");            // si le champ n'est pas vide, on lève une erreur-utilisateur

        return false;

    }

}

# BOM HTML (fenêtre du navigateur)

Le Browser Object Model permet à Javascript d’interagir avec le navigateur utilisé. **Remarque** : Il n’existe pas de normes officielles pour le BOM

## L’objet fenêtre : *window*

window.document.getElementById("header");        // est la même chose que la ligne en dessous

document.getElementById("header");

Remarque : L’objet «window» contient l’objet «document»

## Pop-ups pour l’utilisateur : *window.alert, window.confirm, window.prompt*

window.alert("Les cookies sont désactivés");          // l'utilisateur doit cliquer sur "OK" pour continuer

window.confirm("Voulez-vous activer les cookies ?")   // l'utilisateur doit cliquer sur "OK" (renvoie true) ou "Annuler" (renvoie false)

window.prompt("Quel est votre nom ?", "John Doe")     // "John Doe" est la réponse par défaut, cliquer sur "OK" renvoie la valeur entrée

window.confirm("Activation des cookies \n Voulez-vous activer les cookies ?")   // "\n" pour sauter une ligne dans le texte de la pop-up

## Récupérer la taille de la fenêtre *.innerWidth, .innerHeight*

let w = window.innerWidth;          // récupère la largeur de la fenêtre du navigateur

let l = window.innerHeight;         // récupère la hauteur de la fenêtre du navigateur

## Méthodes de fenêtre *.open(), .close(), .moveTo(), .resizeTo()*

window.open()         // ouvre une nouvelle fenêtre

window.close()        // ferme la fenêtre en cours

window.moveTo()       // déplace la fenêtre en cours

window.resizeTo()     // redimensionne la fenêtre en cours

Propriétés d’emplacement de la fenêtre *.location.href, .location.hostname, .location.protocol*

document.getElementById("info1").innerHTML = window.location.href;       // renvoie l'url de la page en cours

document.getElementById("info2").innerHTML = window.location.hostname;   // renvoie le nom de domaine de l'hébergeur de la page en cours

document.getElementById("info3").innerHTML = window.location.protocol;   // renvoie le protocole utilisé par la page en cours

## Naviguer dans l’historique du navigateur *.history.back(), .history.forward()*

function retourArriere(){

    window.history.back(); // revient sur la page précédente enregistrée dans l’historique

}

<input type="button" value="Retour" onclick="retourArriere()">

function sautAvant(){

    window.history.forward(); // affiche la page suivante enregistrée dans l’historique

}

## Propriétés de l’écran *.width, .heigth, .availWidth, .availHeigth*

document.getElementById("info1").innerHTML = screen.width;               // affiche la largeur totale de l'écran (en pixels)

document.getElementById("info2").innerHTML = screen.height;              // affiche la hauteur totale de l'écran (en pixels)

document.getElementById("info3").innerHTML = screen.availWidth;          // affiche la largeur disponible de l'écran (en pixels)

document.getElementById("info4").innerHTML = screen.availHeight;         // affiche la hauteur disponible de l'écran (en pixels)

# Gestion des cookies

Les cookies sont des données stockées dans de petits fichiers texte sur l’ordinateur de l’utilisateur, qui permettent à une page de se souvenir des informations de l’utilisateur. Les cookies fonctionnent par paire nom-valeur, (par exemple nom d’utilisateur : Louis Cauvet). Ainsi, lorsqu’un navigateur demande une page web à un serveur, les cookies appartenant à la page sont ajoutés à la demande.

## Vérifier l’activation des cookies : *navigator.cookieEnabled*

console.log(navigator.cookieEnabled);           // renvoie "vrai" si les cookies sont activés, "faux" sinon

## Créer un cookie : *document.cookie*

document.cookie = "nom d'utilisateur = Louis Cauvet";         // création du cookie

document.cookie = "nom d'utilisateur = Louis Cauvet; expires=Thu, 29 Dec 2022 12:00:00 UTC";  // on peut ajouter une date d'expiration

document.cookie = "nom d'utilisateur = Louis Cauvet; path=/";         // on peut ajouter le chemin du cookie

**Remarque** : Par défaut, le cookie expire lors de la fermeture de la page.

**Remarque** : Par défaut, le chemin du cookie est celui de la page.

**Remarque** : Pour supprimer un cookie, il suffit de lui affecter une date déjà passée

## Fonction de création de cookie

function nouveauCookie(nomCook, valCook, nbJours) {     // "nomCook" correspond à la clé du cookie, "valCook" à sa valeur

    const d = new Date();

    d.setTime(d.getTime() + (nbJours \* 24 \* 60 \* 60 \* 1000));

    let expiration = "expires="+d.toUTCString();

    document.cookie = nomCook + "=" + valCook + ";" + expiration + ";path=/";

}

nouveauCookie("prénom","Louis",5);                  // le cookie ayant pour clé "prénom" expire au bout de 5 jours

## Récupérer la liste des cookies

let cookies = document.cookie;   // cookies contient une chaine similaire à "cookie1=valeur; cookie2=valeur; cookie3=valeur"

# Le format JSON

JSON est un format de stockage et de transport des données, utilisé par exemple pour envoyer des données d’un serveur à une page web.

**Remarque**: JSON est beaucoup plus simple à analyser que XML , car il peut être directement analysé par une fonction JavaScript standard.

## Syntaxe du JSON

{

  "directeur" :{"age":47, "prénom":"Thierry"},    // objet JSON

"employés":[                             // "employés" est un tableau JSON contenant 3 objets

    {"age":20, "prénom":"Louis"},          // chaque employé est un objet JavaScript, et possède les propriétés "age" et "prénom"

    {"age":20, "prénom":"Zoé"},

    {"age":21, "prénom":"Raphael"}

  ],

  "en faillite" : false,                          // booléen JSON

  "chiffre d’affaires" : null                     // les valeurs nulles sont aussi acceptées

}

**Remarque** : Les objets Date ne sont pas autorisés en JSON, il faut les écrire sous forme de chaine de caractère que l’on pourra reconvertir en date ensuite.

## Convertir un objet JavaScript en JSON : *JSON.stringify()*

const objet = {nom:"Cauvet", naissance:"2002-04-08", ville:"Dijon"};

const objetJson = JSON.stringify(objet);       // "objetJson" vaut {"nom":"Cauvet","naissance":"2002-04-08","ville":"Dijon"}

const tableau = ["Louis","Zoe","Raphael","Alli"];

const tableauJson = JSON.stringify(tableau);

**Remarque** : Comme les dates ne sont pas autorisées au format JSON, la fonction .*stringify()* convertira toutes les dates en chaînes de caractères

## Convertir du JSON en objets JavaScript : *JSON.parse()*

var texteObjet = '{"nom":"Cauvet", "naissance":"2002-04-08", "ville":"Dijon"}'; // littéral d’objet JSON

const objet = JSON.parse(texteObjet);

document.getElementById("personne").innerHTML = objet.nom + "," + objet.ville;     // affiche "Cauvet, Dijon"

var texteTableau = '["Ford", "BMW", "Audi", "Fiat"]'; // littéral de tableau JSON

const tableau = JSON.parse(texteTableau);

document.getElementById("marque2").innerHTML = tableau[1];         // affiche "BMW"

const objet2 = JSON.parse(texteObjet, function (cle, valeur) {     // la fonction en 2ème paramètre permet de parcourir les données

  if(cle == "naissance"){

    return new Date(valeur);        // lorsque l'on analyse la clé "naissance", on effectue la conversion de la valeur en type Date

  } else {

    return valeur;

  }

});

## La méthode JSONP

JSONP est une méthode qui permet d’envoyer des données JSON (comme le permet l’objet *XMLHttpRequest*), mais sans se soucier des problèmes inter-domaines. En effet, la demande de fichier à partir d’un autre domaine n’est pas forcément tout le temps réalisable, alors qu’il n’y a pas de souci pour la demande de script à partit d’un autre domaine. JSONP utilise donc cet avantage en demande des fichiers à l’aide de la balise <script>. Exemple :

**Dans le fichier JavaScript :**

<script src="demo\_jsonp.php"></script>    // demande de récupérer le fichier PHP "demo\_jsonp.php"

function maFonction(objet) {

  document.getElementByI

d("nomObjet").innerHTML = objet.nom;    // affiche le nom de l'objet JSON stocké dans le fichier PHP

}

**Dans le fichier "demo\_jsonp.php" :**

<?php

$myJSON = '{"nom":"Louis", "age":, "ville":"Dijon"}';

echo "maFonction(".$myJSON.");";                       // appelle la fonction "maFonction()" avec les données JSON en paramètres

?>

On peut penser à utiliser une fonction qui ne créée une balise <script> dynamique qu’en cas de besoin, afin d’optimiser les performances du site :

function cliquerBouton() {

  let nvBalise = document.createElement("script");   // crée une nouvelle balise <script> dans la page

  nvBalise.src = "demo\_jsonp.php";                   // ajoute à la balise un attribut "src" qui désigne le fichier associé

  document.body.appendChild(nvBalise);

}

**Remarque** : Si on n’a pas le contrôle sur le fichier serveur, on peut lui indiquer directement en paramètres la fonction à appeler :

function cliquerBouton() {

  let nvBalise = document.createElement("script");       // crée une nouvelle balise <script> dans la page

  nvBalise.src = "demo\_jsonp.php?callback=maFonction";   // on a ajouté dans l'attribut "src" de la balise la fonction de rappel

  document.body.appendChild(nvBalise);

}

# API Web (Application Programming Interface)

Une API Web est une interface de programmation d’application pour le Web. Il existe les API de navigateur, intégrés dans le navigateur afin de faciliter l’accès à certaines données (exemple : API de géolocalisation qui renvoie les coordonnées de l’emplacement du navigateur), et les API tierces qui ne sont pas liées au navigateur et dont il faut télécharger le code pour l’utiliser (ex : API Youtube qui permet d’afficher des vidéos sur un site Web).

## API de validation de formulaire

<input id="id1" type="number" min="0" max="100" required/>  // Ce champ doit être obligatoirement rempli, avec un nombre entre 0 et 100

<button onclick="VerifChamp()">Valider</button>

function VerifChamp() {

    const valid = document.getElementById("id1");

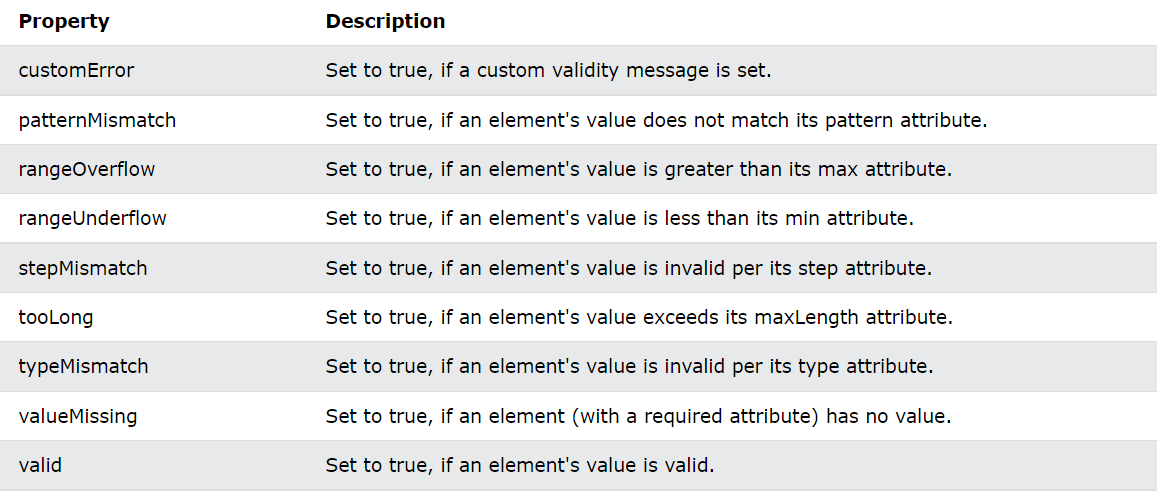
    if(valid.validity.customError){                                          // permet de vérifier si le contenu du champ est valide

        console.log("Il y a une erreur dans la saisie du champ");            // retourne un message de non-validation en cas d'erreur

    }

}

**Remarque**: Il existe plusieurs propriétés de validités :



Retourne «True» si un message de non-validité quelconque est instancié

Retourne «True» si la valeur du champ est plus grande que la limite.

Retourne «True» si la valeur du champ est invalide par son attribut step.

Retourne «True» si la valeur du champ est invalide par son attribut type.

Retourne «True» si la valeur du champ est valide.

Retourne «True» si la valeur du champ ne correspond pas à son attribut pattern.

Retourne «True» si la valeur du champ est plus petite que la limite.

Retourne «True» si la longueur de la valeur est plus grande que son attribut maxLength.

Retourne «True» si un champ obligatoire n’est pas rempli.

## API d’historique

L’objet window.history contient les URL des sites visités par l’utilisateur.

function pagePrecedente(){

    window.history.back();         // charge la dernière page visitée avant celle actuelle (équivaut à la flèche retour du navigateur)

}

function pageSuivante(){

    window.history.forward();      // charge la page suivant l'actuelle (équivaut à la flèche avancer du navigateur)

}

function sautDePage(){

    window.history.go(-3);         // revient 3 pages en arrière par rapport à celle actuelle

}

let l = history.length;            // contient le nombre d'URL stockées dans l'historique

## API de stockage

L’objet localStorage permet de stocker et récupérer des données dans un stockage local associé au domaine d’un site web particulier. Ces données sont stockées sans date d’expiration et ne sont pas supprimées non plus lors de la fermeture du navigateur.

localStorage.setItem("identité", "Louis Cauvet");   // stocke une paire clé-valeur dans la stockage local du site

localStorage.getItem("identité");                   // permet de récupérer la valeur associée à la clé recherchée

localStorage.removeItem("identité");                // supprime l'association liée à la clé recherchée

localStorage.clear();                               // supprime toutes les associations du storage

localStorage.key(2);                                // renvoie la 3ème valeur stockée dans le stockage local

let l = localStorage.length;                        // renvoie le nombre d'objets stockés dans le stockage local

sessionStorage.setItem("identité","Louis Cauvet");  // identique à localStorage, mais supprime les données à la fermeture du navigateur

## API de travail (Web Worker)

Un web worker permet d’effectuer en arrière-plan un code JavaScript sans affecter les performances de la page.

Pour créer un web worker, il faut le définir dans un fichier JavaScript externe. **Exemple :** on crée un fichier demo\_workers.js :

let i = 0;

function compteurTemps(){

    i++;

    postMessage(i);              // postMessage() permet de poster un message sur la page HTML

    setTimeout("timedCount()", 500);

}

compteurTemps();

On crée ensuite un objet de type Worker :

if (typeof(w) == "undefined") {           // vérifie si le worker existe déja ou non

    w = new Worker("demo\_workers.js");    // s'il n'existe pas, on crée un nouveau worker selon le modèle "demo\_workers.js"

}

w.onmessage = function(event){      // "onmessage" est un écouteur d'évènement qui exécute son code lorsque le worker publie un message

    document.getElementById("temps").innerHTML = event.data     // event.data contient les données du web worker

}

function stopWorker() {

    w.terminate();         // termine la mission du worker, qui arrête de s'éxécuter en arrière-plan

    w = undefined;         // rend le worker de nouveau disponible une fois qu'il a été arreté

}

## API de récupération (API Fetch)

L’API Fetch permet au navigateur de faire des requêtes HTTP aux serveurs web afin de récupérer différentes données. Elle est basée sur «async» et «await».

**Exemple :** Récupération d’un fichier et affichage de son contenu

fetch(fichier)                             // équivalent à   async function recupTexte(fichier){

.then(objet => objet.text()) let objet = await fetch(fichier);

.then(texte => myDisplay(texte));     let texte = await objet.text();

myDisplay(texte);

}

## API de géolocalisation

L’API de géolocalisation permet d’obtenir la position géographique d’un utilisateur, à condition que l’utilisateur l’ait approuvé.

**Remarque**: Cette API ne fonctionnera que dans des contextes sécurisés tels que HTTPS.

function getLocation() {

    if (navigator.geolocation) {

      navigator.geolocation.getCurrentPosition(afficherPosition, gererErreurs); // 2ème paramètre=fonction appelée pour gérer les erreurs

    } else {

       console.log("La géolocalisation n'est pas supportée par ce navigateur");

    }

}

function afficherPosition(position) {

    document.write("Latitude: " + position.coords.latitude + "<br>Longitude: " + position.coords.longitude);

}

function gererErreur(erreur) {

    switch(erreur.code) {

      case erreur.PERMISSION\_DENIED:

        console.log("L'utilisateur n'a pas donnée la permission d'accéder à sa localisation");

        break;

      case erreur.POSITION\_UNAVAILABLE:

        console.log("La position n'a pas été retrouvée");

        break;

      case erreur.TIMEOUT:

        console.log("La requête a pris trop de temps pour s'exécuter");

        break;

      case erreur.UNKNOWN\_ERROR:

        console.log("Une erreur inconnue est apparue");

        break;

    }

}

Pour afficher le résultat sur une carte, il faut passer par un service de carte (Google Maps par exemple) :

function montrerPosition(position) {

    let latlon = position.coords.latitude + "," + position.coords.longitude;

    let img\_url = "https://maps.googleapis.com/maps/api/staticmap?center=

    "+latlon+"&zoom=14&size=400x300&sensor=false&key=YOUR\_KEY";

    document.getElementById("carteMaps").innerHTML = "<img src='"+img\_url+"'>";

}

# AJAX

AJAX est une technique permettant d’accéder en arrière-plan à des serveurs Web depuis une page Web. Elle permet notamment la mise à jour de manière asynchrone des pages Web, afin de mettre à jour des parties de la page sans devoir la recharger complètement. Pour cela, elle utilise un objet «*XMLHttpRequest*» intégré au navigateur, qui permet de d’échanger avec un serveur Web.

## Schéma de fonctionnement

Navigateur

Un évènement se produit…

- Créer un objet *XMLHttpRequest*

- Envoyer une *HttpRequest*

Serveur

- Traite la *HttpRequest*

*-* Créer une réponse et la retourne au navigateur

Internet

Internet

Navigateur

- Traite les données récupérées avec JavaScript

- Met à jour la page Web

## Envoyer une demande HttpRequest au serveur Web (GET) *.onload, .open(), .send()*

let xhttp = new XMLHttpRequest();     // créer un nouvel objet XMLHttpRequest

xhttp.onload = function(){            // définir une fonction de rappel à exécuter lorsque la réponse arrive

  // code à exécuter une fois que la réponse est prête

}

xhttp.open("GET", " demo\_get.asp", true);   // le 2ème paramètre correspond au fichier sur le serveur, true indique asynchrone

xhttp.open("GET", "demo\_get.asp?t="+Math.random());  // on ajoute un id unique à l'url pour éviter que le résultat soit mis en cache

xhttp.send();                        // envoyer la requête au serveur

**Remarque :** on utilise «POST» au lieu de «GET» comme 1er paramètre, lorsqu’on envoie une grande quantité de données au serveur ou qu’on lui envoie des entrées-utilisateur.

**Remarque**: La page Web et le fichier chargé (2ème paramètre) doivent se trouver sur le même serveur.

**Remarque**: «true» comme 3ème paramètre indique que la requête s’exécute en arrière-plan (c’est la valeur par défaut, donc pas forcément besoin de l’écrire).

## Envoyer des données au serveur Web (POST) *.setRequestHeader()*

let xhttp = new XMLHttpRequest();     // créer un nouvel objet XMLHttpRequest

xhttp.onload = function(){            // définir une fonction de rappel à exécuter lorsque la réponse arrive

  // code à exécuter une fois que la réponse est prête

}

xhttp.open("POST", "ajax\_test.asp");       // POST pour envoyer des données (dans le cas d'un formulaire HTML par exemple)

xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");   // on ajoute une entête HTTP

xhttp.send("fname=Henry&lname=Ford");      // on spécifie les données que l'on veut envoyer dans la fonction .send()

## Gérer plusieurs tâches AJAX dans un même site web

On doit créer une fonction de rappel pour chaque tâche AJAX :

loadDoc("url-1", maFonction1);

loadDoc("url-2", maFonction2);

function loadDoc(url, fonction){         // fonction d'envoi de la demande

  const xhttp = new XMLHttpRequest();

  xhttp.onload = function() {fonction(this);}

  xhttp.open("GET", url);

  xhttp.send();

}

function maFonction1(xhttp){           // 1ère fonction de rappel

  // code

}

function maFonction2(xhttp){           // 2ème fonction de rappel

  // code

}

## Récupérer la réponse du serveur *.responseText, .responseXML*

Il y a 2 manière de récupérer la réponse du serveur à la requête envoyée :

On peut afficher cette réponse sous forme de chaine de caractères avec *.responseText* :

xhttp.onload = function() {

  document.getElementById("reponse").innerHTML = this.responseText; // affiche les données de la réponse avec une chaine de caractères

}

On peut aussi l’analyser et l’afficher en tant qu’objet DOM XML avec *.responseXML* :

xhttp.onload = function() {

  const xmlDoc = this.responseXML;                           // récupère les données sous forme d'objet DOM XML

  const artistes = xmlDoc.getElementsByTagName("ARTIST");    // récupère les objets des balises "ARTIST"

  let txt = "";                                               // correspond à la liste des artistes qu'on va remplir

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

    txt = txt + artistes[i].childNodes[0].nodeValue + "<br>"; // on remplit la liste donnée par donnée

  }

  document.getElementById("listeArtistes").innerHTML = txt;   // affichage de la liste

}

## Récupérer les informations d’en-tête de la réponse du serveur *.getAllResponseHeaders(), .getResponseHeader()*

On peut récupérer l’ensemble des informations d’en-tête de la réponse avec *.getAllResponseHeaders()*:

Une image contenant texte

Description générée automatiquementxhttp.onload = function() {

  document.getElementById("infos").innerHTML = this.getAllResponseHeaders();

}

Ou alors récupérer une seule information précise avec *.getResponseHeader() ,* et la clé de l’information qui nous intéresse en paramètre:

xhttp.onload = function() {

  document.getElementById("info").innerHTML = this.getResponseHeader("Last-Modified");

}

## Afficher des informations depuis un fichier JSON

const xhttp = new XMLHttpRequest();

xhttp.onload = function() {

  const objet = JSON.parse(this.responseText);

  document.getElementById("personne").innerHTML = objet.nom + "" + objet.prénom;

};

xhttp.open("GET", "fichier\_json.txt");

xhttp.send();

**Remarque** : On peut également récupérer des informations au format JSON contenues dans un fichier PHP (ex : <https://www.w3schools.com/js/js_json_php.asp>).

## Afficher des informations sous forme de tableau à partir d’un fichier XML

function maFonction(xml) {               // cette fonction est appelée en rappel au moment du xhttp.onload

  const xmlDoc = xml.responseXML;        // on récupère la réponse

  const liste = xmlDoc.getElementsByTagName("CD");

  let table="<tr><th>Artist</th><th>Title</th></tr>";    // on indique la structure du tableau

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

    table += "<tr><td>" +

    liste[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue + "</td><td>" +  // on insère les données dans la 1ère colonne

    liste[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue + "</td></tr>";   // on insère les données dans la 2ème colonne

  }

  document.getElementById("tableauCD").innerHTML = table;   // on affiche le tableau

}

## Exemple d’utilisation : Afficher des recommandations lors d’une saisie avec AJAX-PHP

**Dans la partie HTML :**

<p>Commencez à saisir un nom dans le champ de recherche :</p>

<p>Suggestions: <span id="txtHint"></span></p>

<form>

Prénom : <input type="text" onkeyup="analyserSaisie(this.value)">

</form>

**Dans la partie JavaScript :**

function analyserSaisie(saisie) {

  if (saisie.length == 0) {                                  // en cas de champ vide, pas de recommandation

    document.getElementById("txtHint").innerHTML = "";       // on efface le champ des recommandations

    return;

  } else {

    const xmlhttp = new XMLHttpRequest();

    xmlhttp.onload = function() {

      document.getElementById("txtHint").innerHTML = this.responseText;  // indique l'endroit où affcher les recommandations

    }

  xmlhttp.open("GET", "gethint.php?q=" + saisie); // on indique le fichier PHP qui contient les recommandations, avec un paramètre "q" qui vaut la saisie

  xmlhttp.send();

  }

}

**Dans la partie PHP :**

<?php

// tableau des recommandations

$a[] = "Anna";

$a[] = "Brittany";

$a[] = "Cinderella";

$q = $\_REQUEST["q"];  // récupère le paramètre "q" depuis l'URL

$hint = "";

// recherche tous les indices du tableau si $q est différent de "" (champ vide)

if ($q !== "") {

  $q = strtolower($q);

  $len=strlen($q);

  foreach($a as $name) {

    if (stristr($q, substr($name, 0, $len))) {

      if ($hint === "") {

        $hint = $name;

      } else {

        $hint .= ", $name";

      }

    }

  }

}

// Renvoie "no suggestion" si aucune proposition ne correspond à ce qui est saisi

echo $hint === "" ? "no suggestion" : $hint;

?>

**Remarque** : On peut aussi faire de même avec des données stockées dans une base de données, en passant par un fichier PHP qui exécute une requête sur la base et renvoie le résultat dans un tableau HTML (ex : <https://www.w3schools.com/js/js_ajax_database.asp>).

# Export/Import de code

On peut exporter des parties de code (des fonctions par exemple) dans d’autres fichiers JavaScript pour pouvoir les réutiliser en les important lorsque cela est nécessaire. Cela facilite la maintenance de la base de code.

**Remarque :** L’import/export fonctionne uniquement avec le protocole http(s)

## Exporter du code : *export*

**Ex : Dans un fichier personne.js :**

const nom = "Louis";

const age = 20;

export{nom, age}  // le nom et l’âge contenus dans ce fichier peuvent être exportés dans d'autres fichiers

## Importer du code : *import*

**Ex : Dans un fichier message.js :**

import {nom, age} from "./personne.js";

document.write("Je m'appelle " + nom + " et j'ai " + age + " ans."); // Les variables peuvent être utilisées dans ce fichier

# Mode strict

Le mode strict de JavaScript permet d’écrire un code plus propre, à l’aide d’une expression littérale à déclarer en début de script. Ce mode permet d’empêcher d’utiliser des variables non déclarées, de lever des erreurs qui auraient été « mis sous silence » avec le mode normal…

## Déclarer le mode strict

"use strict";         // à placer en début de script

## Quelques règles du mode strict

"use strict";                       // à placer en début de script

x = 3.14;                           // erreur car la variable n'est pas déclarée

y = {p1:10, p2:60};                 // erreur car l'objet y n'est pas déclaré

delete x;                           // erreur car la supression d'une variable n'est pas autorisée

function maFonction(p1, p2){};

delete maFonction();                // erreur car la supression d'une fonction n'est pas autorisée

const obj = {get x() {return 0}};   // erreur car l'écriture dans une propriété en lecture seule n'est pas autorisée

let eval = 3.14;                    // erreur car le mot "eval" ne peut pas être utilisé comme nom de variable

let arguments = 3.14;               // erreur car le mot "arguments" ne peut pas être utilisé comme nom de variable

let public = 3.14;                  // erreur car le mot "public" ne peut pas être utilisé comme nom de variable

let private = 3.14;                 // erreur car le mot "private" ne peut pas être utilisé comme nom de variable

let protected = 3.14;               // erreur car le mot "protected" ne peut pas être utilisé comme nom de variable

let static = 3.14;                  // erreur car le mot "static" ne peut pas être utilisé comme nom de variable

maFonction(x, z);

var z = 5;                          // erreur car une variable ne peut pas être utilisée avant d'être déclarée