I01 – JavaScript

Table des matières

[Présentation 4](#_Toc34232746)

[Références 4](#_Toc34232747)

[Syntaxe et sémantique 5](#_Toc34232748)

[Divers 5](#_Toc34232749)

[Debugger son code 6](#_Toc34232750)

[Les variables 6](#_Toc34232751)

[Opération mathématique 6](#_Toc34232752)

[Les variables textuelles 7](#_Toc34232753)

[Exercice 7](#_Toc34232754)

[Les comparateurs 7](#_Toc34232755)

[Égalité 7](#_Toc34232756)

[Inégalité 8](#_Toc34232757)

[Ternaire 8](#_Toc34232758)

[OR, AND, NOT 8](#_Toc34232759)

[Conditionnel 8](#_Toc34232760)

[Switch 9](#_Toc34232761)

[Type de données 10](#_Toc34232762)

[Les différents types de données en JavaScript 10](#_Toc34232763)

[Vérifier le type d'une variable (typeof) 10](#_Toc34232764)

[Les fonctions 11](#_Toc34232765)

[Opération sur les nombres 12](#_Toc34232766)

[Opérations mathématique de base via Math 12](#_Toc34232767)

[Opération sur les chaines de caractères (string) 13](#_Toc34232768)

[Les fonctions asynchrones 13](#_Toc34232769)

[Documentation des fonctions 14](#_Toc34232770)

[Les tableaux (array) 15](#_Toc34232771)

[Fonction sur les tableaux : 16](#_Toc34232772)

[Les boucles 17](#_Toc34232773)

[While 17](#_Toc34232774)

[For 17](#_Toc34232775)

[Les fonctions de boucle pour les tableaux 18](#_Toc34232776)

[Les objets 18](#_Toc34232777)

[Les classes 19](#_Toc34232778)

[Le DOM : la structure du code HTML en JS 20](#_Toc34232779)

[Les sélecteurs 20](#_Toc34232780)

[Création de nœud et ajout dans le DOM 21](#_Toc34232781)

[Parcourir les nœuds 22](#_Toc34232782)

[Les évènements 22](#_Toc34232783)

[Exemple de type évènements 23](#_Toc34232784)

[Les API JavaScript 23](#_Toc34232785)

[JSON 24](#_Toc34232786)

[Exercice 25](#_Toc34232787)

[Introduction 25](#_Toc34232788)

[Référence 25](#_Toc34232789)

[Plus ou moins 26](#_Toc34232790)

[Références 26](#_Toc34232791)

[Liste de courses 27](#_Toc34232792)

[Gestion de la batterie 28](#_Toc34232793)

[Références 28](#_Toc34232794)

[Puissance 4 29](#_Toc34232795)

[Références 29](#_Toc34232796)

# Présentation

Le JavaScript est un langage de programmation créé en 1995. A la base, il servait principalement à rendre les pages Web dynamiques et interactives. C’est aujourd’hui bien plus que cela, il est maintenant possible de faire des sites web de A à Z, des applications IOS / Android, des applications de bureau, des services Web, des jeux vidéo, …

Un site Web c’est globalement un serveur et un client qui communique (front-end – back-end). Quand un client (utilisateur) demande au serveur une page web, le serveur lui renvoie des informations comme par exemple la page HTML à afficher. Celle-ci est alors censé rester statique jusqu’à ce que l’utilisateur demande le chargement d’une autre page via un hyperlien. Le JavaScript permet de modifier ce fichier HTML, par exemple en fonction des évènements que produit l’utilisateur. Par exemple quand celui-ci passe sa souris au-dessus d’une vignette de vidéo et que celle-ci se lance automatiquement (Netflix ou YouTube). Il permet également de créer des animations, et des jeux vidéo 3D (par exemple : moteur de jeu Unity).

Quelques détails techniques

Comme l’indique son nom, le JavaScript est un langage de script. C’est à dire qu’il a besoin d’un interpréteur pour fonctionner (par exemple le navigateur). Il ne peut pas fonctionner autrement sur votre machine. Ce qui en est autrement pour d’autres langages tels que le C, qui peuvent fonctionner sur votre machine sans que vous n’ayez rien à installer de plus (un fichier .exe par exemple).

Le JS est un langage haut niveau. C’est à dire qu’il fonctionne au plus près de l’utilisateur et plus loin du fonctionnement de l’OS (système d’exploitation). Cela signifie par exemple qu’un script JS ne peut pas intervenir directement sur l’utilisation du processeur, de la gestion de la RAM, … Alors qu’un langage comme le C le permet, et même l’y « oblige ». En effet, le navigateur fait beaucoup pour le programmeur au niveau de la gestion des ressources, là ou en C tout est à faire.

## Références

Tutoriel très complet sur JavaScript (français)

<https://developer.mozilla.org/fr/docs/Web/JavaScript>

Tutoriel complet et à destination des débutant (français)

<https://openclassrooms.com/fr/courses/2984401-apprenez-a-coder-avec-javascript>

Tutoriel qui résume rapidement les différents éléments du JavaScript (anglais facile)

<https://www.w3schools.com/Js/>

Un premier exemple et une excellente page pour tester des petits scripts :

<https://www.w3schools.com/Js/tryit.asp?filename=tryjs_whereto_head>

# Syntaxe et sémantique

La syntaxe d’un langage informatique est la manière dont les différents éléments du langage peuvent être assemblé pour former un programme fonctionnelle (ex : la manière dont une boucle (while, for, …) peut être combinée avec une instruction conditionnelle (if)).

La sémantique est le sens des phrases utilisées dans un langage. Par exemple :

a > b ? a : b ; donne en français : « si a est plus grand que b renvoi a sinon renvoi b ».

Nous allons donc voir le vocabulaire utilisé pour écrire du code JavaScript, la signification et la manière d’agencer les éléments. Notre objectif sera de créer des scripts permettant de réagir aux interactions de l’utilisateur avec le navigateur web et de changer la structure de la page HTML en conséquence.

L’ensemble de la syntaxe du cours est résumé dans un code JavaScript afin de le tester plus facilement et de servir de résumé. Ce fichier est accessible via le repository Git du cours : <https://github.com/martini224/EFP_I01/blob/master/code/syntaxe.html>

## Divers

Écrire un commentaire, c'est à dire du code qui sera ignoré par le navigateur.

// ceci est un commentaire sur une ligne

/\*

Ceci est un bloc commentaire, on peut écrire sur plusieurs lignes.

Les commentaires sont utiles pour attirer l'attention de la personne lisant le code (soi-même ou d’autres personnes) sur un élément qui pourrait être difficile à comprendre, ou simplement pour expliquer le fonctionnement d'un bout de code.

\*/

Il est également possible, voir souhaitable, de documenter son code. Par exemple en décrivant une fonction ou une méthode donnée, le rôle des paramètres d’entrée et qu’est-ce que la fonction va renvoyer.

Documenter et commenter son code sont deux choses primordiales pour s’assurer que son code puisse être lisible et compréhensible par tous et même soi-même. Cela permet de travailler plus facilement en équipe et de déboguer plus efficacement du code. Plus d’explication sur la document dans le chapitre sur les fonctions.

### Debugger son code

Un outil très utile pour tester et déboguer son code et outils développeur présent sur la plupart des navigateurs. Pour ouvrir l’ouvrir, il suffit d'appuyer sur F12 sur votre navigateur préféré. Vous y trouverez notamment la console ou sera affiché différents messages d'erreur ou de log. Par exemple si vous écrivez console.log("bonjour"); dans votre code JS, bonjour s'affichera dans la console.

Plus d'info sur la console et le débogage :

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Introduction#La_console_web>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/debugger>

## Les variables

Une variable permet de stocker une donnée ou un ensemble de données pouvant être traité.

var nomDeLaVariable; déclaration d'une variable. Une variable doit être déclarée avant d'être utilisée.

Le nom d'une variable en JS ne doit pas commencer par un chiffre et ne doit pas contenir de caractères spéciaux (ex : é, @, ù, ...).

Il est préférable que le nom commence par une minuscule et que chaque autre mot que compose le nom de la variable commence par une majuscule. Il est préférable que ne nom de la variable soit explicite quant à son utilité.

### Opération mathématique

nomDeLaVariable = 3; on assigne la valeur entière 3 à la variable

nomDeLaVariable += 3; on ajoute 3 à la valeur de la variable (3+3, donc 6)

nomDeLaVariable = nomDeLaVariable + 3; écriture plus longue mais équivalente.

let n = 0.5; Une variable peut être déclarée avec le mot clé "var" ou "let". On préférera utilisé "let".

n = n + 3 - 3 + (2 \* (5 / 2)); un exemple d’opération mathématique écrite en JS.

\*\* = exponentiel

% modulo, c'est à dire le reste du résultat de la division. Exemple 11%2 = 1 car 11 = 5 x 2 + 1.

n++; on incrémente de 1 la valeur de la variable x, ce qui est équivalent à x += 1 ou x = x + 1.

n--; on décrémente de 1 la valeur de la variable x, ce qui est équivalent à x -= 1 ou x = x - 1.

### Les variables textuelles

let texte = "text"; on assigne une valeur textuelle à la variable texte. Une valeur ou une variable de type textuelle est appelée un string.

texte = 'text'; on réassigne une nouvelle valeur à cette variable.

Remarquez qu'une valeur textuelle (un string) peut s'écrire entre " " ou entre ' '.

texte = 'bonjour' + ' ' + "le monde"; permet de fusionner plusieurs string, on appelle cela la concaténation.

texte.length; renvoie la longueur de la chaine de caractères

### Exercice

Quel sera le résultat de :

let x = 5 + 5;

let y = "5" + 5;

let z = "Hello" + 5;

let y = "5" - 5;

## Les comparateurs

let a = 1;

let b = 2;

let aText = '1';

### Égalité

a == b; renvoie false

a === b; renvoie false

a == aText; renvoie true

a === aText; renvoie false

### Inégalité

a != b; renvoie true

a !== b; renvoie true

a != aText; renvoie false

a !== aText; renvoie true

a < b; renvoie true

a <= b; renvoie true

a <= aText; renvoie true

### Ternaire

a > b ? "si true alors on affiche ceci" : "si false on affiche cela"; renvoie si false…

### OR, AND, NOT

a < b && a == aText; renvoie true. && = ET (AND) ce qui veut dire en français : si a est plus petit que b ET que a égal aText. Les deux conditions doivent être vraies.

a < b || a == aText; renvoie true. || = OU (OR° ce qui veut dire en français : si a est plus petit que b OU que a égal aText. Seule une seule condition vraie suffit.

!(a < b); a < b = true et le ! (NOT) devant signifie l'inverse du résultat, donc renvoie false

### Conditionnel

if(a < b) {

// la condition est vraie.

alert("Vrai");

}

Else {

// la condition est fausse.

alert("Faux");

}

### Switch

Un switch permet de soulager l'écriture des conditionnelles IF quand il y a beaucoup de cas possible.

Dans le code suivant, new Date() ; permet d'obtenir la date du jour, et la fonction new Date() .getDay() ; permet d'obtenir le numéro du jour de la semaine.

switch (new Date().getDay()) {

case 0: // La semaine calendrier commence le dimanche en JS

day = "Dimanche";

break;

case 1:

day = "Lundi";

break;

case 2:

day = "Mardi";

break;

case 3:

day = "Mercredi";

break;

case 4:

day = "Thursday";

break;

case 5:

day = "Friday";

break;

case 6:

day = "Saturday";

break;

}

console.log(day); va afficher le nom du jour

Pour en savoir plus sur les dates :

<https://www.w3schools.com/js/js_dates.asp>

## Type de données

Dans les langages de programmation, les variables sont typées. C’est-à-dire qu’elle corresponde à un certain type de données, par exemple un nombre entier positif, un nombre réel, une chaine de caractères, un objet, … C’est le cas en JavaScript où il y a 6 types de base. Le JS est dit à typage dynamique, car le type d’une variable déclarée peut changer en cours d’exécution d’un code JS (un nombre entier peut devenir un string pendant l’exécution du programme).

Un exemple de langage n’ayant pas un typage dynamique est le Java, où le typage est dit statique. Une variable y est déclarée avec son type (ex : int unEntier ; String uneChaineDeCaracteres ;) et la variable doit toujours avoir une valeur de ce type. Dans un code Java où une variable string déclarée comme ceci : String prenom = "Martin"; ne pourra avoir comme instruction suivante : prénom = 1; car la variable prenom est une chaine de caractères ce qui est incompatible avec un entier.

En JavaScript, on n’indique pas de type au moment de la déclaration de la variable puisque le typage y est dynamique. Si let prénom = "Martin"; on pourra écrire comme instruction suivante : prénom = 1; Ce mécanisme entraine donc parfois la conversion implicite des données. Par exemple l’instruction suivante en JS : "5" - 5; donnera comme résultat 0 car le string "5" est automatiquement considéré comme un nombre par le JS, étant donné la présence du signe moins. Mais dans le cas d’une addition (+) le JS va interprété le signe comme étant le symbole de concaténation de deux string. Donc le résultat de "5" + 5 ; donne le string "55".

### Les différents types de données en JavaScript

**string** pour une chaine de caractères.

**number** un nombre 1, 1202015, -1, -5.21245.

**boolean** true, false (vrai ou faux).

**undefined** type non défini qui indique que la variable n'est pas définie.

**function** pour une fonction.

**object** pour un objet.

### Vérifier le type d'une variable (typeof)

let myVar;

typeof myVar; undefined

myVar = 1;

typeof myVar; number

myVar = function(i){

return i++;

} f

typeof myVar; function

myVar = {nom: "Martin", prenom: "Rouffiange"};

typeof myVar; object

# Les fonctions

Une fonction est un bloc de code exécutant une tâche bien précise. Elle peut avoir (et ce de manière générale en informatique) des valeurs d'entrée (que l'on appel arguments ou paramètres de la fonction) et peut retourner un résultat (la valeur de retour). Les fonctions sont particulièrement utiles pour découper le code en plusieurs sous tâches afin de le rendre plus lisible et de pouvoir réutiliser facilement le code afin d’éviter la duplication de code (du code qui fait plusieurs fois la même chose).

Par exemple une fonction de conversion qui convertit les degrés fahrenheit en degrés Celsius :

function toCelsius(fahrenheit) {

return (5 / 9) \* (fahrenheit - 32);

}

toCelsius(10); renvoie -12.222222222222223

Il est en effet plus facile d'écrire le calcul une fois pour toute que de le répéter plusieurs fois (même si le calcul n’est fait qu’une fois, il est toujours bon de « l’isoler » dans une fonction. Ceci afin de pouvoir le réutiliser quand on le souhaite, mais aussi de pouvoir retrouver le calcul plus facilement dans le code et de rendre l’utilité du calcul plus intelligible (grâce au nom de la fonction par exemple).

A chaque fois que vous avec le sentiment que vous allez utiliser plusieurs fois un même bout de code, ou que vous avez déjà écrit un bout de code quelque part ou encore qu'un bout de code commence à devenir trop volumineux, c'est qu'il faut découper votre code en plus petites fonction. Seul l'expérience pourra vous donnez le bon feeling à ce niveau. Mais la règle est que l'informaticien est paresseux, ce qui signifie qu'il n'aime pas faire quelque chose qui a déjà été fait et qu'il pourrait réutiliser (que ce soit par lui-même ou par quelqu'un d'autre).

Il existe un tas de fonctions déjà écrite pour nous. Par exemple Bootstrap qui permet de mettre en forme le HTML plus facilement. On peut alors récupérer ce code et l'ajouter au notre si la licence de celui-ci en donne la permission. Il existe par ailleurs beaucoup de fonctions de base en JS.

### Opération sur les nombres

isNaN permet de vérifier si une variable n'est pas un nombre (is Not a Number)

isNaN(10); renvoie false car 10 est un nombre

isNaN("x"); renvoie true car x n'est pas un nombre

Number("10"); crée un nombre à partir d'un nombre ou d'un string

Number("10.33"); crée un nombre à partir d'un nombre ou d'un string qui peut contenir une virgule

parseInt("10"); crée un nombre à partir d'un nombre ou d'un string qui ne peut pas contenir une virgule (nombre entier)

parseFloat("10.33"); crée un nombre à partir d'un nombre ou d'un string qui peut contenir une virgule

### Opérations mathématique de base via Math

Arrondi supérieur

Math.round(4.7); renvoie 5

Puissance : ici 8 puissance 2

Math.pow(8, 2); renvoie 64

Racine carrée

Math.sqrt(64); renvoie 8

Arrondi supérieur

Math.ceil(4.4); renvoie 5

Arrondi inférieur

Math.floor(4.7); renvoie 4

Généré un nombre réel pseudo-aléatoire entre 0 (inclus) et 1 (exclus)

Math.random();

Généré un nombre entier (grâce à Math.floor()) pseudo-aléatoire entre 0 (inclus) et 1 (exclus)

Math.floor((Math.random() \* 10) + 1);

Plus de fonctions mathématique : <https://www.w3schools.com/js/js_math.asp>

En savoir plus sur les nombres pseudo-aléatoires :

<https://fr.wikipedia.org/wiki/Générateur_de_nombres_pseudo-aléatoires>

### Opération sur les chaines de caractères (string)

let str = "Apple, Banana, Kiwi";

substring : renvoie la chaine de caractères situé entre le 7ème caractère et le 13ème (inclus)

str.substring(7, 13); renvoie un nouveau string 'Banana'

replace : Remplacement du 1er élément correspondant au premier paramètre de la fonction

str.replace('Banana', 'Banane') ; renvoie un nouveau string 'Apple, Banane, Kiwi'

replace avec une expression régulière (regex) : Remplacement de tous les éléments correspondant au premier paramètre de la fonction

str = 'pomme pomme';

str.replace(/pomme/g, 'poire') ; renvoie un nouveau string 'poire poire'

toUpperCase : Mettre la chaine de caractères en majuscule

str.toUpperCase(); renvoie "POMME POMME"

toLowerCase : Mettre la chaine de caractères en minuscule

str.toLowerCase(); renvoie "pomme pomme"

trim : Supprime les espaces du début et de la fin de la chaine de caractères

str = ' pomme ';

str.trim(); renvoie "pomme"

split : sépare la chaine de caractères en un tableau de string. La séparation se fait via la chaine de caractères passée en paramètre

str = 'pomme pomme pomme';

str.split(" "); renvoie un tableau ['pomme', 'pomme', 'pomme']

Plus d’informations à propos des fonctions sur les chaines de caractères :

<https://www.w3schools.com/js/js_string_methods.asp>

### Les fonctions asynchrones

Dans le fonctionnement normal de l’exécution du code JavaScript, chaque instruction est exécutée dans l’ordre d’écriture, et l’instruction suivante n’est exécutée qu’une fois l’exécution précédente terminée.

Exemple :

demanderMotDePasse() ;

véridierMotDePasse() ;

envoyerReponse() ;

On demande le mot de passe, puis on vérifie celui-ci pour finalement envoyer la réponse appropriée. Chaque instruction s’exécute dans l’ordre et ce en attendant que la précédent soit terminée.

Une fonction asynchrone permet de ne pas respecter cette « liste » d’attende. Ainsi dans une fonction asynchrone, les instructions peuvent ne pas attendre que la précédente soit terminée pour s’exécuter.

Pour déclarer une fonction asynchrone, il suffit de mettre le mot clé async devant la déclaration d’une fonction :

async function uneFonctionAsynchrone() {

telechargerFilm() ;

afficherBarreChargement() ;

}

Plus d’informations sur les fonctions asynchrones :

<https://openclassrooms.com/fr/courses/5543061-ecrivez-du-javascript-pour-le-web/5577651-comprenez-comment-fonctionne-lasynchrone-en-js>

### Documentation des fonctions

Ecrire de la documentation, c’est donné la description d’une fonction ou une méthode donnée, de ses paramètres d’entrée et sa valeur de retour. La documentation permet également de décrire beaucoup plus, par exemple les préconditions et postconditions.

Les préconditions sont des conditions qui s’applique aux paramètres de la fonction et qui doivent être vrai avant l’exécution de la fonction (par exemple une chaine de caractères non vide).

Les postconditions sont des conditions qui s’applique à la valeur de retour d’une fonction et qui doivent être vrai après exécution de celle-ci (la valeur de retour doit être un entier positif).

Elles permettent de s’assurer que ce que fait la fonction est bien ce à quoi on s’attend. Cela est donc très utile pour déboguer son code. Nous ne les utiliserons néanmoins pas dans ce cours.

Voici un exemple de documentation pour la fonction de conversion fahrenheit / Celsius :

/\*\*

\* Convertit une température reçue en degrés fahrenheit en degré Celsius.

\*

\* @param farenheit la température en degrés fahrenheit

\* @return la température en degrés Celcius

\*/

function toCelsius(fahrenheit) {

return (5 / 9) \* (fahrenheit - 32);

}

## Les tableaux (array)

Les tableaux permettent de stocker plusieurs données en même temps, typiquement une liste d’objet (une liste de joueurs, liste d'items...)

Exemple de déclaration d'un tableau avec des valeurs string

let names = ["Martin", "Jérome", "Henri"];

Déclaration d'un tableau vide

names = [];

Ecritures équivalentes :

names = new Array();

names = new Array("Martin", "Jérome", "Henri");

Accéder à un élément du tableau :

names[0]; renvoie Martin

Connaitre le nombre d'éléments dans le tableau :

names.length; renvoie 3

Si on essaye d'accéder à un élément qui n'est pas dans le tableau :

names[3]; renvoie undefined

Il est donc toujours préférable de vérifier la longueur d'un tableau avant d'accéder à un élément :

if(names.length > 3){

names[3];

} else {

/\*

La taille du tableau est = 3, car il y a 3 élément. Or on essaye d'accéder au 4ème élément, ce qui ne peut pas fonctionner.

En effet, le premier élément du tableau est accessible via names[0]; Il faut donc faire -1 pour accéder à l'élément souhaité.

Ex : le 4ème élément : 4-1 = 3

\*/

}

### Fonction sur les tableaux :

Join : permet de fusionner les éléments du tableau dans un string.

let fruits = ["Banane", "Orange", "Pomme", "Mangue"];

fruits.join(" / "); renvoie un string : "Banane / Orange / Pomme / Mangue"

shift : retire le premier élément du tableau

fruits.shift(); retire "Banane"

pop : retire le dernier élément du tableau

fruits.pop(); retire "Mangue"

unshift : ajoute un élément au début du tableau

fruits.unshift("Citron"); ajoute citron au tableau

push : ajoute un élément à la fin tableau

fruits.push("Citron"); ajoute orange au tableau

concat fusionne deux tableaux :

let names2 = ["Robin", "Morgan"];

names.concat(names); renvoie un nouveau tableau ["Martin", "Jérome", "Henri", "Robin", "Morgan"]

sort : trie le contenu du tableau dans l'ordre croissant

names2.sort(); le tableau names2 devient : ["Morgan", "Robin"]

reverse : trie le contenu du tableau dans l'ordre décroissant

names2.reverse(); le tableau names2 devient : ["Robin", "Morgan"]

Plus de fonctions sur les tableaux :

<https://www.w3schools.com/Js/js_array_methods.asp>

## Les boucles

Une boucle permet d'exécuter le même code plusieurs fois, tant qu'une condition est vraie.

### While

let i = 0;

while(i < 10) {

console.log(i); // affiche 0, 1, 2, .. jusqu’à 9 dans la console

i++;

}

### For

Une autre façon d'écrire une boucle plus condensée :

for(let j = 0; j < 10; j++) {

console.log(j); // affiche 0, 1, 2, .. jusqu'à 9 dans la console

}

Les boucles sont particulièrement utiles pour traiter le contenu d’un tableau :

fruits = ["Banane", "Orange", "Pomme", "Mangue"];

for(iFruit = 0; iFruit < fruits.length; iFruit++) {

console.log(fruits[iFruit]); // affiche chaque élément du tableau dans la console

}

Une autre façon de traiter un tableau plus condensée :

for(iFruit in fruits) {

console.log(fruits[iFruit]); // affiche chaque élément du tableau dans la console

}

### Les fonctions de boucle pour les tableaux

Les tableaux possèdent une fonction propre pour parcourir chaque élément :

fruits.forEach(fruit => console.log(fruit)); affiche chaque élément du tableau dans la console

Les tableaux possèdent également d’autres fonctions pour traiter leurs éléments :

map : permet de renvoyer une copie du tableau a laquelle on a appliqué un traitement

let numbers = [45, 4, 9, 16, 25];

numbers.map(number => number \* 2); renvoie une copie du tableau 90,8,18,32,50 (on a fait x2 à chaque élément)

filter : renvoie une copie du tableau où chaque élément répond à une condition donnée

numbers = [45, 4, 9, 16, 25];

numbers.filter(number => number > 18); renvoie une copie du tableau 45,25 (les éléments plus grand que 18)

Plus de fonctions d’itération des tableaux :

<https://www.w3schools.com/Js/js_array_iteration.asp>

## Les objets

Un objet permet de stocker plusieurs variables pouvant avoir des types différents, dont des fonctions. Un objet sert à représenter un concept, une entité du monde réel. Par exemple une voiture, un joueur, une personne, … En JavaScript les tableaux sont d’ailleurs des sortes d’objets.

Tous les langages de permettent pas la manipulation d’objet (pas d’objet en C, en Java oui). Les langages de programmation manipulant des objets s’appel des langages orienté objet. On parle également de programmation orientée objet. Un tel paradigme implique de nombreux concepts que nous ne ferons qu’effleurer dans ce cours.

Par exemple un objet représentant un joueur :

let object\_player = {nom: 'jouer', level: 3};

### Les classes

Les informations sur les classes si dessous sont à titre informatif dans le cadre de ce cours.

Les variables d'une classe sont appelé ses attributs, et les fonctions les méthodes. Les classes sont une manière de représenter facilement ce à quoi doit ressembler un objet, c'est à dire ce que doit être ses attributs et méthodes (ex : une voiture possède des roues, des portes, un volant, …).

class Player {

constructor(name, level){

this.name = name;

this.level = level;

}

displayName(){

console.log(this.name);

}

}

Quand on crée un objet d'une certaine classe, on dit qu'on instancie la classe.

object\_player = new Player('joueur', 3); quand on affiche la variable object\_player dans la console, cela ressemblera a : {nom: 'joueur', level: 3}

object\_player.displayName(); fait appel à la méthode displayName qui affiche le nom du joueur dans la console

object\_player.name; accède à l'attribut name qui renvoie le nom du joueur

Pour en savoir plus sur les classes : <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes>

## Le DOM : la structure du code HTML en JS

Document Object Model ou DOM (pour modèle objet de document), est la représentation en arbre du document HTML.

Le DOM est composé de nœuds, qui sont des objets ayant des attributs et des méthodes permettant de connaitre et modifier le document HTML. La représentation en arbre permet de voyager dans le document, exemple : parent <- nœud -> enfants, frères. Tout nœud (sauf la racine = document) possède un parent, et peut avoir un ou plusieurs enfants.

Un nœud peut être un élément HTML, un texte ou un commentaire, mais seul les nœuds HTML peuvent avoir des enfants (bien que certaines balises HTML ne soit pas prévue pour, par exemple <option>).

### Les sélecteurs

Sélecteurs : ils permettent de sélectionner un élément dans le DOM

document.getElementById('id'); renvoie l'élément possédant l'id passé en paramètre

document.getElementsByClassName('classe'); renvoie un tableau d'éléments possédant la classe passée en paramètre

document.querySelector('#id'); renvoie le premier élément correspond au sélecteur CSS passé en paramètre

document.querySelectorAll('.classe'); renvoie un tableau d'éléments correspond au sélecteur CSS passé en paramètre

La liste des sélecteurs CSS :

<https://www.w3schools.com/cssref/css_selectors.asp>

let element = document.querySelector('#demo');

querySelector() et querySelectorAll() peuvent également être utilisé sur un élément précis, la recherche ce fera alors parmi toutes la descendance de l’élément.

element.querySelector('#id');

Afficher ou modifier le contenu textuel d'un élément

element.innerText; pour l'afficher

element.innerText = "test"; pour le modifier

Afficher ou modifier le code HTML d'un élément

element.innerHTML; pour l'afficher

element.innerHTML = "<div><p></p></div>"; pour le modifier

Accéder à un attribut (les attributs HTML : id, name, class, type, …)

element.getAttribute('id'); renvoie l'id de l'élément, ici 'demo'

Afficher ou modifier le code CSS d'un élément (remplace tout le code)

element.style.cssText = "color: blue; border: 1px solid black";

Une autre manière de l'écrire :

element.setAttribute("style", "color:red; border: 1px solid blue;");

Affiche, modifie ou ajoute un élément CSS précis, en laissant les autres intactes.

element.style.color = "blue";

Afficher, gérer les classes d'un élément :

element.classList; renvoie les classes de l'élément, ici 'demo\_classe'

Ajouter une classe :

element.classList.add('une\_classe');

Retirer une classe :

element.classList.remove('une\_classe');

Si la classe est présente la retire, si elle est absente l'ajoute :

element.classList.toggle('une\_classe');

Renvoie true si l'élément possède la classe :

element.classList.contains('une\_classe');

### Création de nœud et ajout dans le DOM

Création d'un nouvel élément, ici un div

let div = document.createElement('div');

On ajoute cet élément à l'élément body

document.querySelector('body').appendChild(div);

div.innerText = 'Nouvel élément';

### Parcourir les nœuds

element.parentElement; renvoie le parent de l'élément

element.previousSibling; renvoie le frère précédent l'élément (que ce soit un élément, texte ou commentaire)

element.previousElementSibling; renvoie le frère précédent l'élément (uniquement élément)

element.nextSibling; renvoie le frère suivant l'élément (que ce soit un élément, texte ou commentaire)

element.nextElementSibling; renvoie le frère suivant l'élément (uniquement élément)

element.childNodes; renvoie les enfants de l'élément (que ce soit un élément, texte ou commentaire)

element.children; renvoie les enfants de l'élément (uniquement élément)

## Les évènements

Les évènements permettent de réagir par rapport aux évènements produits par l'utilisateur, c'est à dire les actions qu’il produit sur la page HTML. Par exemple cliquer sur un bouton, passer sa souris au-dessus d'un élément, ... Il est possible d’écouter un évènement à partir du document ou par rapport à un élément précis en faisant appel à la fonction addEventListener. On lui passera alors en paramètre le type d’évènement à écouter et la fonction à exécuter une fois que l’évènement se produit.

let boutton = document.querySelector('#btn');

Deux façons équivalente de l’écrire :

boutton.addEventListener('click', reaction);

boutton.addEventListener('click', function(){

// Le contenu de la fonction

});

function reaction() {

// Le contenu de la fonction

}

### Exemple de type évènements

|  |  |
| --- | --- |
| click | Un bouton d'un dispositif de pointage a été appuyé ou relâché sur un élément. |
| dblclick | Un bouton d'un dispositif de pointage a été double-cliqué sur un élément. |
| contextmenu | clique droit de la souris |
| focus | Un élément a acquis le focus. |
| focusout | Un élément perd le focus. |
| fullscreenchange | Un élément est passé en mode plein écran, ou est revenu au mode normal. |
| keydown | Une touche est relâchée. |
| keyup | Une touche est enfoncée. |
| keypress | Une touche est pressée et cette touche produit normalement un caractère (utilisez input à la place). |
| change | Un élément perd le focus et sa valeur a changé depuis l'acquisition du focus. |
| chargingchange | La batterie commence ou finit de charger. |

La liste des types d'évènements :

<https://developer.mozilla.org/fr/docs/Web/Events>

## Les API JavaScript

Une API (interface de programmation d’application) est un ensemble de classes, méthodes, fonctions et constantes qui offre des services à un logiciel. Nativement, le JS offre des fonctionnalités que vous pouvez utiliser pour accéder à des éléments de votre navigateur par exemple. Elles permettent entre autres de gérer les cookies, utiliser la webcam, géolocaliser l’utilisateur, traduire des textes, …

Utiliser la webcam <https://developer.mozilla.org/fr/docs/WebRTC/Prendre_des_photos_avec_la_webcam>

Géolocalisation

<https://developer.mozilla.org/fr/docs/Web/API/Geolocation_API>

Liste des API JS

<https://developer.mozilla.org/fr/docs/Web/API>

## JSON

Le JSON (JavaScript Object Notation) est une syntaxe pour sérialiser des objets, tableaux, nombres, chaînes de caractères, booléens. C’est-à-dire transformer un élément JavaScript en une chaine de caractères le décrivant.

Le JSON n’est pas uniquement utilisé en JavaScript, mais qui sert aussi à échanger des données entre deux services Web par exemple, et qui ne sont pas nécessairement écrit en JS. Un exemple d’utilisation, est le code JavaScript appelant un service météo (via une API) qui va renvoyer la météo en fonction de la géolocalisation qu’on lui a indiqué sous format JSON. Les informations vont ensuite est traitée et affichée de manière plus lisible à l’utilisateur grâce à l’HTML et CSS.

Soit la chaine de caractère JSON suivante :

let json = '{"nom":"martin", "age":25, "skills":["JavaScript", "HTML", "CSS"]}';

Celle-ci peut être transformée en un objet JavaScript grâce à JSON.parse() :

let objectFromJson = JSON.parse(json);

Et ce même objet peut être transformé à son tour en un string JSON :

json = JSON.stringify(objectFromJson);

Voici un exemple de JSON qui sera utilisé dans les exercices ci-après :

<https://github.com/martini224/EFP_I01/blob/master/code/js/liste_courses.json>

# Exercice

## Introduction

Dans ce chapitre est repris quelques exercices afin de pratiquer le JavaScript. La seule manière efficace d’apprendre et de s’exercer. Diverses compétences seront développées dans ces exercices : compréhension d’un énoncé, conversion de l’énoncé en une esquisse de la tâche à accomplir (comment résoudre le problème), ensuite développement de la solution. Pour chaque exercice, un début de code est disponible sur le repository git : <https://github.com/martini224/EFP_I01/tree/master/code>

Vous pouvez partir de là pour réaliser les exercices. Ils vous permettront principalement de vous concentrer sur la partie JavaScript plutôt que HTML / CSS. Des solutions aux exercices seront déposées sur le repository <https://github.com/martini224/EFP_I01_correction>. Ne les attendez pas pour faire les exercices / surmonter une difficulté.

Quand vous ne savez pas comment faire quelques choses, relisez le cours et / ou faites une recherche sur un moteur de recherche en énonçant clairement votre problème. Beaucoup de personnes avant vous se sont surement posée les mêmes questions. N’hésitez donc pas à consulter des sites comme <https://stackoverflow.com/> où vous pourrez voir les questions déjà posée ou en posée vous-mêmes et d’autres utilisateurs vous répondrons sans doutes.

### Référence

Documentation bootstrap (mise en page, secondaire dans ce cours)

<https://getbootstrap.com/docs/4.4/getting-started/introduction/>

Liste des icônes avec fontawesome (de nouveau secondaire dans ce cours)

<https://fontawesome.com/icons?d=gallery&m=free>

## Plus ou moins

Nous allons créer un jeu qui fait deviner un nombre à l’utilisateur. Le jeu génère un nombre aléatoire entre 1 et 100, et l’utilisateur doit le deviner en 10 tours maximum. Pour chaque tentative, le jeu dit si l’estimation de l’utilisateur est trop ou pas assez élevée.

Penser comme un programmeur

Une des choses les plus difficiles à apprendre en programmation n'est pas la syntaxe, mais comment l'appliquer afin de résoudre un problème réel. Vous devez commencer à penser comme un programmeur — ce qui implique généralement d'examiner les tâches que votre programme doit effectuer, de déterminer les fonctionnalités du code nécessaires à leurs réalisations et comment les faire fonctionner ensemble.

Cela requiert un mélange de travail acharné, d'expérience avec la syntaxe de programmation (de manière générale) et surtout de la pratique — ainsi qu'un peu de créativité. Plus vous allez coder, plus vous aller vous améliorer.

Voici comment on peut découper le problème :

1. Générer un nombre aléatoire entre 1 et 100.
2. Stocker le nombre d’essais restant. Commencer par 10.
3. Fournir au joueur le moyen de saisir un nombre.
4. Stocker l'ensemble des propositions de nombres pour que le joueur puisse les consulter.
5. Vérifier si le nombre saisi par le joueur est correct.
6. S'il est correct :
   * 1. Afficher un message de félicitations.
     2. Empêcher que le joueur saisisse de nouveau un nombre.
     3. Afficher un contrôle pour que le joueur puisse rejouer.
7. S'il est faux et que le joueur a encore des tours à jouer :
   * 1. Informer le joueur que sa proposition de nombre est fausse.
     2. Lui permettre d'entrer une nouvelle proposition de nombre.
     3. Décrémenter le nombre d’essais de 1.
8. S'il est faux et que le joueur n'a plus de tours à jouer :
   * 1. Informer le joueur qu'il a perdu et que la partie est finie.
     2. Empêcher que le joueur saisisse de nouveau un nombre.
     3. Afficher un contrôle pour que le joueur puisse rejouer.
9. Une fois le jeu redémarré, s'assurer que la logique du jeu et l'interface utilisateur sont complètement réinitialisées, puis revenir à l'étape 1.

### Références

Cet exercice est repris du site mozzila.org <https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/A_first_splash>

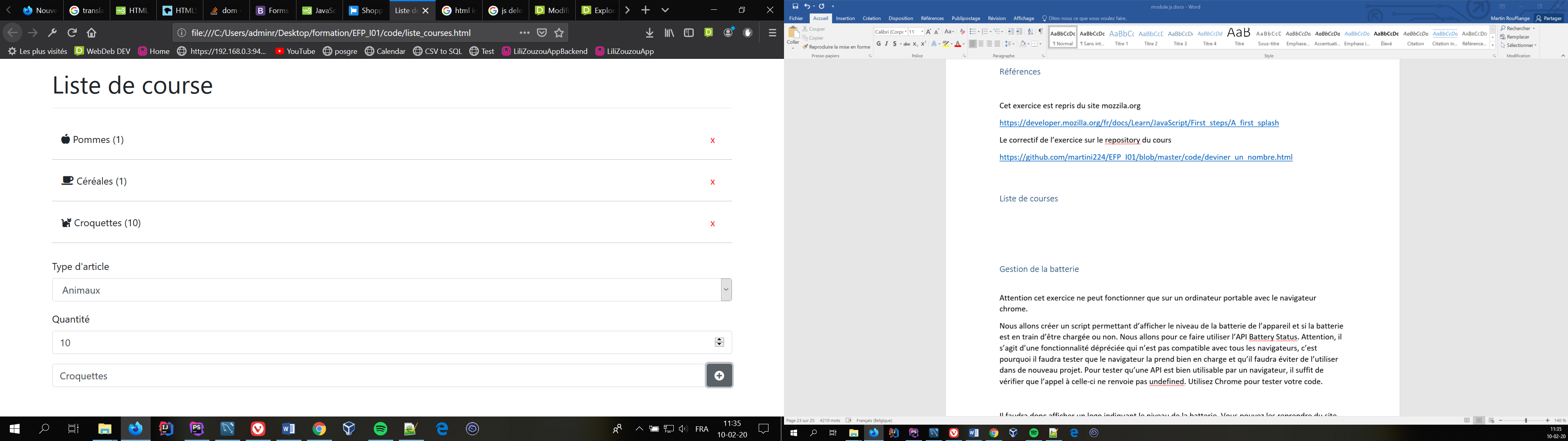
## Liste de courses

Le but de cet exercice sera la création d’une liste des courses à partir d’un formulaire où l’on pourra indiquer le type de l’article à acheter, la quantité et l’intitulé de celui-ci. L’utilisateur se verra proposé une liste de proposition d’intitulé d’article quand il commencera a tapé dans le champ de l’intitulé.

Cette liste est fournie au format JSON, pour chaque catégorie est renseigné un id technique, son intitulé et une liste d’intitulé d’articles faisant partie de la catégorie. Quand aucun type d’article n’est sélectionné, il doit être proposé à l’utilisateur les propositions contenant les lettres qu’il a tapé dans le champ intitulé, et ce pour n’importe quelle catégorie. Quand une catégorie est indiquée, il ne doit alors être proposé que les articles de cette catégorie.

L’utilisateur doit avoir la possibilité d’ajouté un article à la liste via un bouton, et de supprimé un article de la liste via un autre bouton.

Le résultat final pourrait ressembler à la figure ci-dessous :



Attention, le JavaScript ne peut pas lire de fichier présent sur l’appareil de l’utilisateur pour des raisons de sécurité. Il ne vous sera donc pas possible de lire le fichier liste\_courses.json. Il est sur repository à titre d’exemple de ce à quoi un tel fichier peut ressembler. Celui-ci à été ‘minimifier’ en une ligne dans le fichier liste\_courses.js. Considérez donc la constante jsonText comme étant la valeur du fichier liste\_courses.json.

## Gestion de la batterie

Attention cet exercice ne peut fonctionner que sur un ordinateur portable avec le navigateur chrome.

Nous allons créer un script permettant d’afficher le niveau de la batterie de l’appareil et si la batterie est en train d’être chargée ou non. Nous allons pour ce faire utiliser l’API Battery Status. Attention, il s’agit d’une fonctionnalité dépréciée qui n’est pas compatible avec tous les navigateurs, c’est pourquoi il faudra tester que le navigateur la prend bien en charge et qu’il faudra éviter de l’utiliser dans de nouveau projet. Pour tester qu’une API est bien utilisable par un navigateur, il suffit de vérifier que l’appel à celle-ci ne renvoie pas undefined. Utilisez Chrome pour tester votre code.

Il faudra donc afficher un logo indiquant le niveau de la batterie. Vous pouvez les reprendre du site fontawesome : <https://fontawesome.com/icons?d=gallery&q=battery&m=free>. Si le niveau de batterie est < 25% on affiche le logo battery-empty, si < 50 : battery-quarter, si < 75 battery-half, si < 100 battery-three-quarter et si = 100 battery-full. Vous pouvez également attribuer une couleur différente en fonction du niveau de batterie.

En plus de cela, il faudra écrire le niveau de batterie en pourcent et afficher un logo (par exemple : <https://fontawesome.com/icons/bolt?style=solid> ) quand la batterie est en charge.

Ces données devront être affichée dès le lancement de la page HTML sur lequel votre script va fonctionner, mais également quand l’évènement chargingchange se produit (c’est-à-dire que le chargeur vient d’être branché ou débranché) et l’évènement levelchange quand le niveau de la batterie change.

### Références

L’API Battery Status

<https://developer.mozilla.org/fr/docs/Web/API/Battery_Status_API>

## Puissance 4

L’objectif sera de réaliser un puissance 4. Il s’agit d’un exercice pour aller plus loin, qui vous demandera beaucoup de temps (si vous faites des animations, …) et vous permettra de grandement améliorer votre maitrise du JavaScript.

Une solution sera ajoutée au repository <https://github.com/martini224/EFP_I01_correction>, n’hésitez pas à la consulter.

### Références

Les règles du puissance 4 :

<https://fr.wikipedia.org/wiki/Puissance_4>