JAVASCRIPT

Partie 1 - Initiation

QU'EST-CE QUE LE JAVASCRIPT?

Le JavaScript est un langage de programmation de scripts orienté objet.

Il permet :

- D'améliorer l'esthétique d'un logiciel ou site Internet
- D'améliorer l'ergonomie d'un site Web
- De mettre à jour ou faire apparaitre des éléments sans devoir recharger la page
- De réagir d'une manière spécifique aux actions de l'utilisateur.

QU'EST-CE QUE LE JAVASCRIPT?

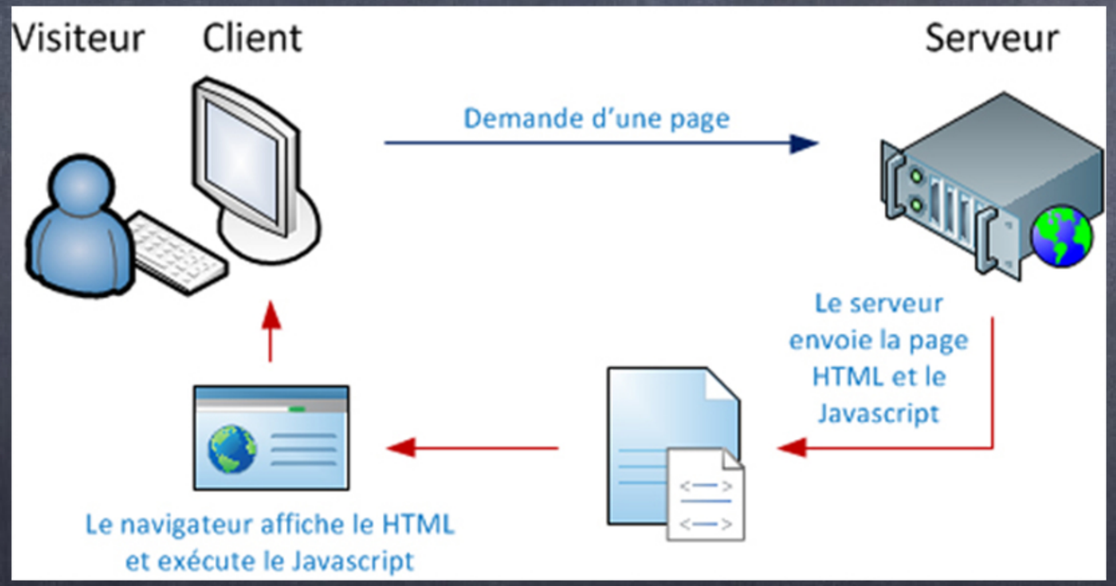
Il s'agit d'un langage interpreté : Le code n'est pas compilé.

Si on veut l'exécuter, il faut faire appel à un interpréteur qui se chargera de l'analyser et de réaliser les actions qu'il contient. (Exemple Ignition Chez google pour le moteur V8JS)

Remarque: pour améliorer les performances, la plupart de ces interpréteurs procèdent à une pseudo précompilation "à la volée" pendant l'exécution (si le même code doit être réexécuté, il sera beaucoup plus rapide)

QU'EST-CE QUE LE JAVASCRIPT?

Le JavaScript est un langage dit client-side.
Les scripts sont exécutés par le navigateur à la différence de PHP ou les scripts sont exécutés par le serveur.



QU'EST-CE QUE LE JAVASCRIPT?

Le javascript a été initialement conçu pour être utilisé avec le HTML pour de l'amélioration esthétique.
Depuis quelques années d'autres utilisations :

-Extension de navigateur
-Utilisation avec node.js (JS coté serveur)
-Création d'applications

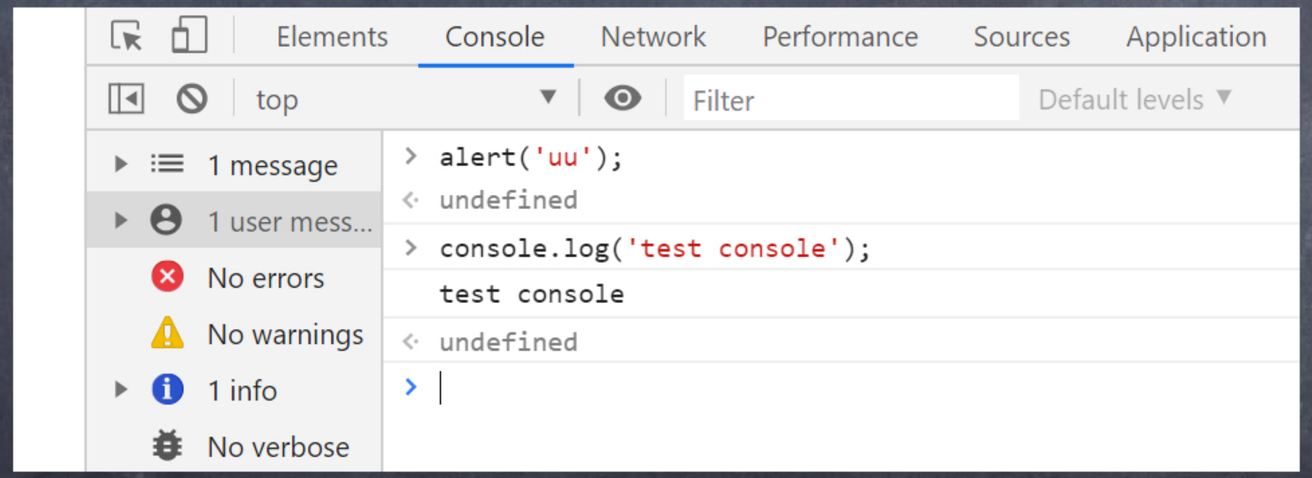
Ne pas confondre Java et Javascript se sont des langages totalements différents.
Version JS actuelle ES6 (2014) supportée par les navigateurs récents mais moins bien par les plus anciens - Solution Babel

LE PREMIER SCRIPT

```
<script type="text/javascript">
    alert('Bonjour à tous'); //AFFICHE UNE BOITE DE DIALOGUE
    console.log('utilisation de la console.'); //AFFICHE EN CONSOLE
</script>
```

Affiche une pop-up avec votre texte - ALERT

Affiche du texte dans la console - CONSOLE.LOG (F12 - chrome)

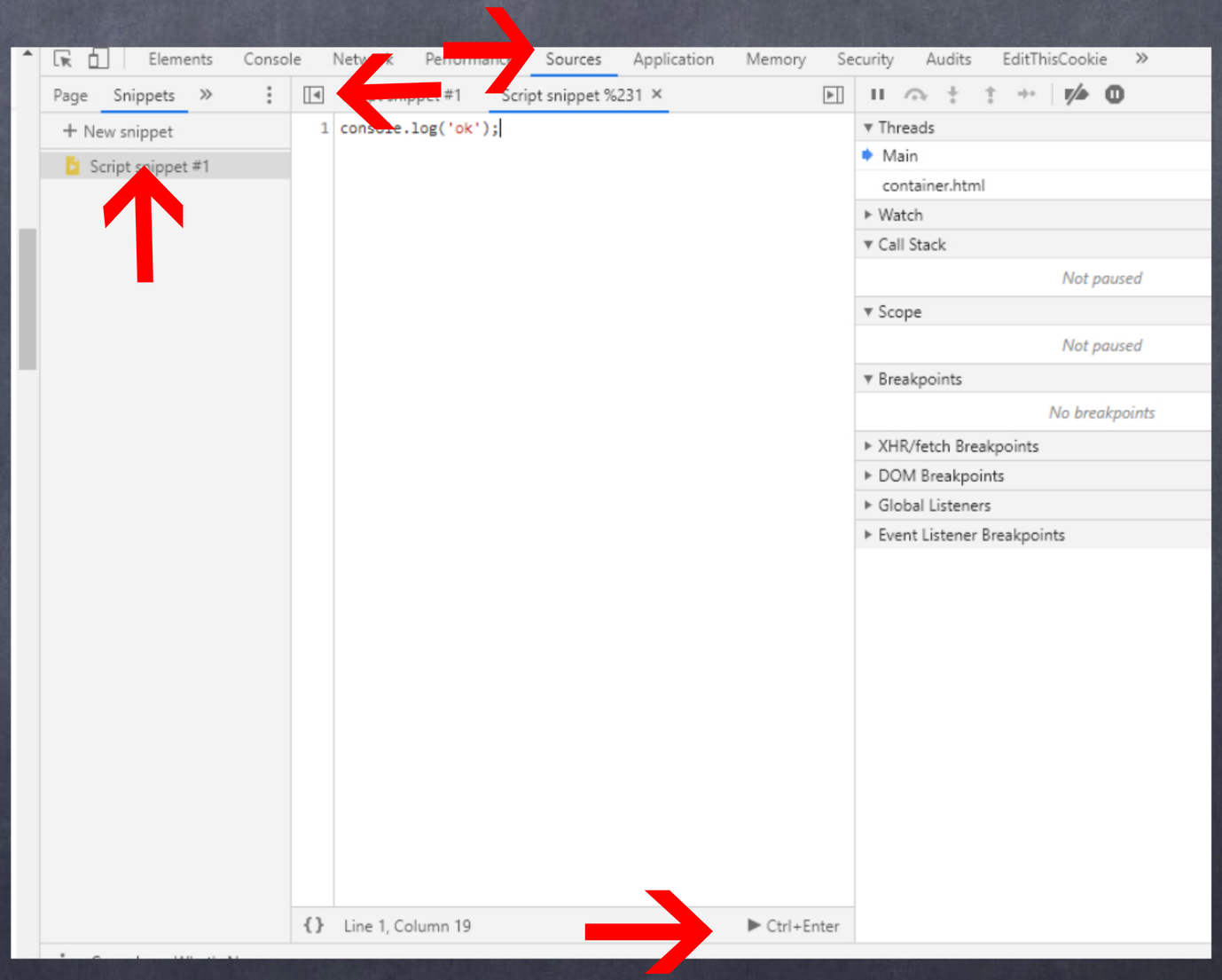


Vous pouvez écrire directement ce code dans la console

Il sera interprété

JAVASCRIPT DANS CHROME

1. Ouvrir outils développement (F12)
2. Onglet sources
3. Cliquez sur le chevron
4. Selectionnez snippet
5. New snippet
6. Inserez votre code
7. Ctrl + enter ou cliquez en bas



SCRIPT EXTERNE

Créez un fichier exemple1.js

0-first.php ● | 1-externe.php | JS exemple1.js × | index.html

javascript-initiation > assets > js > JS exemple1.js

```
1   alert('Bonjour à tous'); //AFFICHE UNE BOITE DE DIALOGUE
2   console.log('utilisation de la console.'); //AFFICHE EN CONSOLE
```

On appelle le script externe - A placer en bas du body pour eviter de bloquer le rendu HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script type="text/javascript" src="assets/js/exemple1.js"></script>
</body>
</html>
```

VARIABLE

ES5 : VAR

```
var x = "externe";

function test(interne) {
if (interne) {
    var x = "interne"; // à la fonction entière comme portée
    return x;
}
return x; // est redéfini parce que la définition en ligne 4 est hissée (hoisted)
}
console.log(test(false),test(true)); // undefined
; // "interne"
```

VAR à la fonction pour portée.
La variable est disponible dans toute la fonction, même avant d'être déclarée
Les déclarations sont hissées, vous permettant d'utiliser une variable avant sa déclaration
Les initialisations ne sont pas hissées. Si vous utilisez var, déclarez toujours vos variables avant le reste

ES6 : Let et CONST

```
let z = "externe";
function test2(interne) {
if (interne) {
    let z = "interne";
    return z;
}
return z; // on obtient le résultat de la ligne 1, comme attendu
}
test2(false); // "externe"
test2(true); // "interne"
```

Let comportement plus naturel

Const est utilisé pour une variable dont la valeur ne change pas et également pour la déclaration de fonction.

```
const testvar = "Bonjour";
console.log(testvar);
testvar = "Oui"; //ERREUR ON NE PEUT REDEFINIR UNE CONSTANTE
```

CHAINE DE CARACTERES

Manipulation de chaines de caractères

```
let chaineGuillement    = "Je programme en Javascript"; //GUILLEMET OU QUOTE
let chaineQuote         = 'Je programme en Javascript';
console.log(chaineGuillement,chaineQuote);
let escape = 'Je m\'appelle Sébastien'; //ANTISLASH POUR ECHAPPER CARACTERE
console.log(escape);
let calcul = 'Ceci est un test';
console.log(calcul.length);//NOMBRE DE CARACTERES
console.log(calcul.charAt(3)); //RETOUR LETTRE VIA INDICE
console.log(escape.concat(', ', chaineQuote)); //CONCATENATION
```

Je programme en Javascript Je programme en Javascript	3-chaines.php:12
Je m'appelle Sébastien	3-chaines.php:14
16	3-chaines.php:16
i	3-chaines.php:17
Je m'appelle Sébastien, Je programme en Javascript	3-chaines.php:18

Modèle de libellés - ES6 permet l'utilisation de l'interpolation.

Il est important d'utiliser des «ticks arrières» (CTRL+ALT+7 deux fois)

```
//ES5
var prenom = "Jean", nom = "Dupond";
console.log("Vous vous appelez " + prenom + " " + nom + ".");

//ES6 - Utilisation de Babel pour être compatible IE
const prenom2 = "Julie", nom2 = "Machu";
console.log(`Vous vous appelez ${prenom2} ${nom2}.`);
```

CONDITION

Les opérateurs de comparaison

==	égal à
!=	différent de
===	contenu et type égal à
!==	contenu ou type différent de
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

Les opérateurs logiques

&&	ET	valeur1 && valeur2
||	OU	valeur1 || valeur2
!	NON	!valeur

CONDITION SI

If / Else

(Prompt permet de choisir une valeur)

```
//SI CONDITION
var info = parseInt(prompt('Choisir un valeur numérique :'));
if(info <= 20) {
    alert('Plus petit que 20');
} else {
    alert ('Plus grand que 20');
}
```

IF version confirm

(Boite de dialogue)

```
//ALTERNATIVE IF AVEC CONFIRM
if(confirm('Acceptez la demande?')){
    alert('ok');
} else {
    alert('Pas Ok');
}
```

Tester l'existance du variable

En dessous version ternaire (raccourci)

```
//TESTER EXISTANCE VARIABLE
if(info) { alert('existe');} else { alert('existe pas');}
//VERSION TERNAIRE
(info)?alert('existe'):alert('existe pas');
```

CONDITION SWITCH

Switch pour éviter l'imbrication de if/else

```
//SWITCH
var info2 = parseInt(prompt('Choisir un valeur numérique :'));
switch(info2){
    case 1:
        console.log('1 valide');
        break;
    case 2:
        console.log('2 valide');
        break;
    default:
        console.log('Non valide');
}
```

INCREMENTATION

Incrémentation simple et incrémentation raccourci

```
var chiffre = 0;
chiffre = chiffre + 1;
console.log(chiffre);
chiffre += 1;
console.log(chiffre);
```

Attention à l'ordre de l'incrémentation

A noter : La décrémentation fonctionne de la même façon

```
var chiffre2 = 0;
var sortie2 = ++chiffre2;
console.log("chiffre :",chiffre2,"sortie :",sortie2);
```

Chiffre : 1 sortie : 1

```
var chiffre3 = 0;
var sortie3 = chiffre3++;
console.log("chiffre :",chiffre3,"sortie :",sortie3);
```

Chiffre : 1 sortie : 0

BOUCLE WHILE

While permet de réaliser des instructions jusqu'au remplissage d'une contrainte

```
var nombre = 0;
while(nombre<=20){
    nombre = nombre + 1;
    console.log(nombre);
}
```

Do / While : très proche du while
permet d'exécuter au moins une fois un bloc d'instructions
même si la contrainte n'est pas remplie

```
var info = 0;
do{
    info++;
    console.log(info);
}while(info >= 10);
```

Exercice :

Créer un script qui vous demande via une boite de dialogue une chaine de caractère ou un chiffre. le script se stop au moment ou vous fournissez une chaine vide.

Vous afficherez le resultat contenant l'ensemble des caractères saisie via la console.(A faire avec While et do/while)

BOUCLE FOR

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

La boucle for permet de boucler des instructions
3 informations pouvant être fournies
- Initialisation
- Condition
- Incrémentation

```
for (var total = '', nom; true;) {
    nom = prompt('Entrez une valeur :');

    if (nom) {
        total += nom + ' ';
    } else {
        break;
    }
}

console.log(total);
```

Exemple plus complexe
1: initialisation de 2 variables
2: condition «true» toujours vraie
3: Aucune incrémentation
On sort du «for» grâce au «break».

FONCTION

```
function premiere(){
    console.log('ma première fonction');
}
premiere();
```

```
function calcul(){
    var nombre = parseInt(prompt('Donnez une valeur numérique :'));
    console.log("Valeur" + nombre + " *2 = " + nombre*2)
}
calcul();
```

Création de la première fonction :
- Mot clé : function devant le nom de la fonction. Le nom de la fonction ne peux être celui d'une variable
- Définir les instructions
- Appel de la fonction

Attention à la portée des variables

```
//PORTEE VARIABLE
var message = 'Ici la variable globale !';

function infomessage() {
    var message = 'Ici la variable locale !';
    console.log("Dans la fonction " + message);
}
infomessage();
console.log("En dehors " +message);
```

FONCTION

```
function information(arg1,arg2,arg3) {
    if (typeof arg3 === 'undefined') {
        console.log("Arg3 n'a pas été défini");
    }
    console.log(arg1,arg2,arg3);
}
```

Arguments de fonction :
Les arguments sont à spécifier dans le prototype de la fonction.
Les arguments facultatifs sont à placer à la fin.

En cas de non utilisation, il renvoie «undefined»
Vous pouvez tester une variable avec «typeof»
Typeof (number, string, boolean, undefined, function..)

```
function retour(){
    return 'ok';
    console.log('affichage');
}
```

Valeurs de retour:
L'instruction return met fin à la fonction et renvoie la valeur défini (texte, variable)
Ce qui se trouve après n'est pas traité.

FONCTION ANONYMES

```
var test = function(arguments,a,definir){
    console.log(arguments);
}

test("bonjour M2I");
```

Les fonctions anonymes ne possèdent pas de nom
Elles s'utilisent de la même manière

```
var test = 'Oui';
(function() {
    var test = 'Non';
    console.log(test);
})();

console.log(test);
```

IIFE : Immediately Invoked Function Expression
Isolation du code
Permet une plus grand souplesse et une meilleure méthodologie sur la portée des variables avec VAR
ES6 : On préférera l'utilisation de LET et CONST

FONCTION ANONYMES

Cas d'utilisation : Modèle de module code isolé avec fonction publique / private

```
var myModule = (function() {

function privateFunction() { console.log("I can't be accessed from outside :("); }

var privateVar = "me too :(";

function publicFunction() { console.log("Hey! I'm here!"); }

/* expose what you want */
return {
    publicFunction2: publicFunction,
    publicFunction: publicFunction
}

})();

myModule.publicFunction2(); //Hey! I'm here!
myModule.publicFunction(); //Hey! I'm here!
myModule.privateFunction(); //undefined;
```

FONCTIONS FLÉCHÉES (ES6 UNIQUEMENT)

Utilisation de babel standalone pour IE
Si simple «return» pas d'accolade
Si un seul argument pas besoin de parenthèse
Ne pas utiliser les fonctions fléchées en méthode d'objet car pas de notions de «this».

```
<script src="node_modules/@babel/polyfill/dist/polyfill.min.js"></script>
<script src="node_modules/@babel/standalone/babel.min.js"></script>

<script type="text/babel">
//EMULATION - mode document 11 agent 8
let somme = function(a, b) {
    return a + b;
};

let somme2 = (a, b) => a + b;

alert(somme(1, 2));
alert(somme2(1, 2));

let somme3 = a => a+3;
alert(somme3(5));

</script>
```

PARAMÈTRES PAS DEFAUT (ES6)

ES6 facilite grandement les choses

ES5

```
function point(x, y, estMarque){
    x = x || 0;
    y = typeof(y) === "undefined" ? -1 : y;
    isFlag = typeof(estMarque) === 'undefined' ? true : estMarque;
    console.log(x,y, isFlag);
}

    point(0, 0); // 0 0 true
    point(0, 0, false); // 0 0 false
    point(1);// 1 -1 true
    point(); // 0 -1 true
```

ES6 (Babel!!)

```
function point2(x = 0, y = -1, estMarque = true){
    console.log(x, y, estMarque);
}

point2(0, 0); // 0 0 true
point2(0, 0, false); // 0 0 false
point2(1); // 1 -1 true
point2(); // 0 -1 true
```

PAUSE BABEL!

Babel est un transpileur de code, il permet de transformer du javascript dit «moderne» dans une version plus ancienne afin d'être interprété par des navigateurs plus anciens.

Cela est très utile pour permettre d'avoir une compatibilité plus large pour il existe plusieurs moyens de l'utiliser (via des CDN, via des tasks builder type webpack ou via CLI, nous utiliserons CLI :

-Installer node JS : https://nodejs.org/en/ pour disposer du gestionnaire de dépendance NPM
-Rendez vous sur https://babeljs.io/setup#installation et choisissez «CLI»
-Dans votre nouveau projet, initalisez le terminal en utilisant
«npm init» : Cela permet d'initialiser un projet utilisant NPM
«npm install --save-dev @babel/core @babel/cli» : cela permet de telecharger les dependances babel et babel cli
-Ensuite créé un dossier src ou vous mettrez votre code JS
-Modifiez le package.json pour ajouter ce script : "scripts": {"build": "babel src -d lib"}
-Lancer la commande «npm run build»

PAUSE BABEL!

A gauche mon fichier ES6, à droite la transpilation de babel.
Cela nécessite un peu de configuration mais l'ensemble se fait ensuite automatiquement!

index.html | test.js | package.json | babel.confi ...

src > test.js > ...

```
1  [1, 2, 3].map(n => n + 1);
```

test.js src | test.js lib

lib > test.js > ...

```
1  "use strict";
2
3  [1, 2, 3].map(function (n) {
4    return n + 1;
5  });
```

OBJET / TABLEAU

Toute variables JS est en réalité un objet.
Pour les nombres / textes ce sont des objets natifs.

Le JavaScript est un langage orienté objet par prototype. En modifiant les prototypes d'une classe cela modifie l'ensemble des objets appartenant à la classe.

On ne modifie jamais les prototypes des objets natifs.

CONTENU D'UN OBJET JS

CONTRUCTEUR : Le code initial lancé lors de l'instanciation d'un objet JS
PRORIETE : Variable(s) contenu dans l'objet, portée limité à l'objet
METHODE : Fonction(s) contenu dans l'objet, portée limité à l'objet

OBJET / TABLEAU

Exemple rapide :

```
<script>
var nombre = 14; // On crée un objet number
console.log(nombre.toString()); // On utilise une méthode de l'objet nombre
console.log(nombre.toExponential(2)); // On utilise une méthode de l'objet nombre

var texte = "BonJour A Tous!!!";
console.log(texte.length); // On utilise une méthode de l'objet texte
console.log(texte.slice(3,8)); //On utilise une méthode de l'objet texte
console.log(texte.toLowerCase());//On utilise une méthode de l'objet texte
</script>
```

Les objets natifs (String, number, bolean, array) possèdent de nombreuses méthodes utilent.
Visualisez en le comportement en recherchant par exemple :
«number methods Javascript» dans google , le site W3schools est beaucoup moins verbeux et plus compréhensible.

OBJET / TABLEAU

Instanciation tableau (objet de type tableau)

```
var tab1 = ['pomme', 14, 'oui', true]; //Version Courte
var tab2 = new Array('pomme', 14, 'oui', true); //Version Longue
console.log(tab1,tab2);
```

Récupérer une valeur d'un tableau

```
console.log(tab1[0],tab2[3],tab2[15]);
```

L'indice d'indexation d'un tableau commence à 0;
Si on essaye d'affiche une valeur non existante, pas d'erreurs
mais affichage d'un «undefined»

Modifier une valeur d'un tableau

```
tab1[0] = "ok";
console.log(tab1[0]);
```

OBJET / TABLEAU

Instanciation tableau (objet de type tableau)

```
var tab1 = ['pomme', 14, 'oui', true]; //Version Courte
var tab2 = new Array('pomme', 14, 'oui', true); //Version Longue
console.log(tab1,tab2);
```

Récupérer une valeur d'un tableau

```
console.log(tab1[0],tab2[3],tab2[15]);
```

L'indice d'indexation d'un tableau commence à 0;
Si on essaye d'affiche une valeur non existante, pas d'erreurs
mais affichage d'un «undefined»

Modifier une valeur d'un tableau

```
tab1[0] = "ok";
console.log(tab1[0]);
```

MANIPULATION TABLEAU

Méthodes utiles pour manipuler un tableau

```
//AJOUT TABLEAU FIN
var tableau = [1,2];
tableau.push(3);
tableau.push(4,['a','b','c']);
console.log('ajout fin:',tableau);

//AJOUT DEBUT TABLEAU
tableau.unshift('ajout','au','debut');
console.log('ajout debut :',tableau);

//SUPPRESSION PREMIER ELEMENT
tableau.shift();
console.log('suppression premier element:',tableau);

//SUPPRESSION DERNIER ELEMENT
tableau.pop();
console.log('suppression dernier element:',tableau);
```

PARCOURIR UN TABLEAU

En utilisant const on ne peut redéfinir une variable mais on peut manipuler un tableau (push,pop,etc...)

```
// for
var tableau = ["a", "b", "c", "d"];
for (var i = 0; i < tableau.length; i++) {
var element = tableau[i];
console.log(element);
}

// forEach Attention non comptatible <ie9
tableau.forEach(function (element) {
console.log(element);
});

//ES6 seulement
const tableau2 = ["a", "b", "c", "d"];
for (const element of tableau2) {
    console.log(element);
}
```

OBJETS LITTERAUX / TABLEAU ASSOCIATIF

Les objets litteraux sont des «objets temporaires» et non une classe.(Bien comprendre).

Ils ne sont pas réutilisable.(On ne les instancie pas)

Ils sont moins utilisés que des objets standards.

Ils peuvent servir de tableau associatif, retour de fonction, objets transitoires.

Ils contiennent des variables, tableaux, fonctions.

Attention un objet litteral n'est pas un flux JSON même si la syntaxe est quasi-similaire.

OBJETS LITTERAUX / TABLEAU ASSOCIATIF

Les objets litteraux sont des «objets temporaires» et non une classe.(Bien comprendre).

Ils ne sont pas réutilisable.(On ne les instancie pas)

Ils sont moins utilisés que des objets standards.

Ils peuvent servir de tableau associatif, retour de fonction, objets transitoires.

Ils contiennent des variables, tableaux, fonctions.

Attention un objet litteral n'est pas un flux JSON même si la syntaxe est quasi-similaire.

OBJETS LITTERAUX / TABLEAU ASSOCIATIF

```
var first = {
    nom : 'Stone',
    prenom : 'Emma',
    age: 31,
    pays: 'Etats-unis',
}
```

Accès aux attributs

```
//ACCES ATTRIBUTS
console.log(first.nom,first['nom']);
```

Ajout attributs

```
//AJOUT ATTRIBUTS
first['info'] = 'Celibataire';
//OU
first.info = 'En couple';

console.log(first.info);
```

OBJETS LITTERAUX / TABLEAU ASSOCIATIF

Parcourir un objet littéral

```
for(var aux in first) {
    console.log(first[aux]);
}
```

On utilisera for/in pour parcourir un objet litteral
aux est un variable qui reçoit les identifiants défini dans l'objet

EXEMPLE PLUS COMPLEXE

```
var litteral = {
    nom : 'Dupond',
    prenom : 'Jean',
    age : '30',
    hobbies : ['foot','jeux vidéo','cuisine'],

    info : function() {
        console.log(this.nom + ' ' + this.prenom + ' à ' + this.age + ' aime le ' + this.hobbies[0] );
    }
}

console.log(litteral.nom,litteral.prenom);
litteral.info();
```

Les objets littéraux sont souvent utilisés en retour de fonction pour renvoyer plusieurs valeurs voir des fonctions

ES5 / ES6

Obtenir éléments tableau

```
//ES5
var tableau = [1, 2, 3, 4],
premier = tableau[0],
troisieme = tableau[2];

console.log(premier, troisieme); // 1 3

//ES6
const tableau = [1, 2, 3, 4],
  [premier, ,troisieme] = tableau;
console.log(premier, troisieme); // 1 3
```

Echanger des valeurs

```
//ES5
var a = 1, b = 2;
var tmp = a;
a = b;
b = tmp;
console.log(a, b); // 2 1

//ES6
let c = 1, d = 2;
[c, d] = [d, c];

console.log(c, d); // 2 1
```

ES5 / ES6

Décomposition

```
function marges() {
var gauche=1, droite=2, haut=3, bas=4;
return { gauche: gauche, droite: droite, haut: haut, bas: bas };
}

var data = marges(),
    gauche = data.gauche,
    bas = data.bas;
console.log(gauche, bas); // 1 4


function marges2() {
const gauche2=1, droite2=2, haut2=3, bas2=4;
return { gauche1: gauche2, droite1: droite2, haut1: haut2, bas1: bas2 };
}
const { gauche1, bas1 } = marges2();
console.log(gauche1, bas1); // 1 4
```

EXERCICE

En reprennant l'exercice réalisé pour la boucle while

```
var texte , final = "", final2 = "";

do{
    texte = prompt('Insérez une donnée');
    final += texte + " ";
}while(texte!="");

console.log(final);
alert('methode2');
while(1){
    texte = prompt('Insérez une donnée');
    if(texte) {
        final2 += texte + " ";
    } else {
        break;
    }
}
console.log(final2);
```

Stocker les valeurs dans un tableau
A la fin les afficher sous une chaine de caractère séparé par une virgule.

Si le tableau est vide, afficher que un message l'indiquant

Vous aurez besoin de la méthode «push», «console.log», «join» (permet de transformer un tableau en chaine), «length» nombre d'éléments dans un tableau.

CORRECTION

```
var tableau_result = [];
var chaine;

while(chaine = prompt('Insérez une donnée')){
    tableau_result.push(chaine);
}

if(tableau_result.length){
    console.log(tableau_result.join(","));
} else {
    console.log('vide');
}
```

IMPORT EXPORT

Concept venant de ES6 permettant de hierarchiser son projet.
Il est possible d'utiliser babel pour simuler le comportement sur des navigateurs ne prenant pas en charge cette fonctionnalité (par exemple certains navigateurs sur android mobile)
Attention ne fonctionne pas en local donc a utiliser via (liveserver par exemple)
Nous pouvons exporter des variables, constantes, fonctions, objets.
Il s'agit de l'import export natif ES6 ne pas confondre avec import/export de webpack

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="src/export.js" type="module"></script>
    <script src="src/import.js" type="module"></script>
</body>
</html>
```

Index.html

```
rc > JS export.js > ...
1  function bonjour(){
2      console.log('ok');
3  }
4
5  let info = "Bonjour";
6  const VALEUR = 8;
7
8  export {bonjour,info,VALEUR};
```

export.js

```
src > JS import.js
1
2  import {bonjour,VALEUR} from "./export.js";
3
4  console.log(VALEUR);
```

import.js