JAVASCRIPT

LIEN JAVASCRIPT-HTML

Le lien javascript s'inscrit dans le code html avant la fermeture de <body>

```
<script src="./script.js"></script>
</body>
```

SYNTAXE

1. COMMENTAIRES :

//xxx ⇒ Commente une ligne
/*xxx*/ ⇒ Commente tout ce qui est compris entre /* et */

2. INSTRUCTIONS :

Se terminent par ; tlifa
ex : alert("xxx"); ⇒ fait apparaître un pop up avec le texte xxx

3. RÈGLES DE NOMMAGE :

Utilisation du camelCase (très) fortement recommandée

4. ORGANISATION DU CODE :

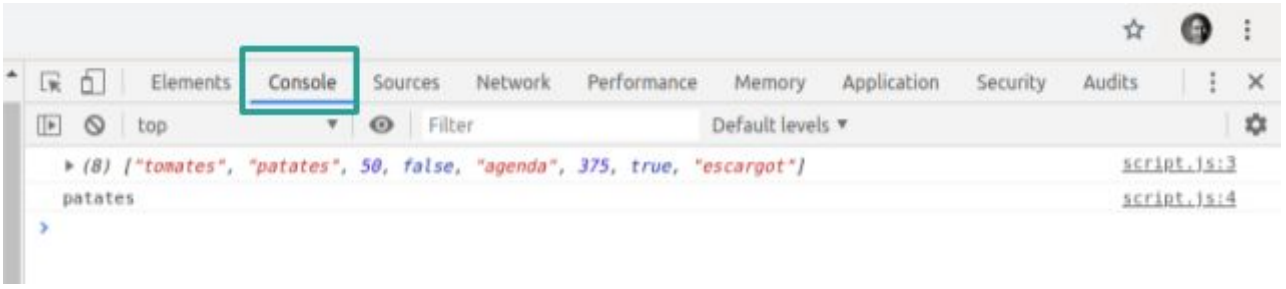
1-constantes
2-variables normales
3-Fonctions rangées par thèmes et documentées

4-code

CONSOLE

1. ACCÈS :

Sur la page web, clic droit, inspecter, console :



2. INSTRUCTION :

console.log(instruction); ⇒ Affiche le résultat de l'instruction dans la console

3. RÈGLES D'UTILISATION :

Les console.log servent pendant le développement, elles doivent être retirées avant la mise en prod.

4. MESSAGES D'ERREUR :

Ils apparaissent en rouge dans la console avec une indication sur le type d'erreur et son lieu (ligne) dans le code

VARIABLES

Typage :

Le type (string, number, function…) d'un élément se retrouve grâce à typeof
ex : typeof(élément);

1. VARIABLES STANDARDS

Déclaration :

let nomDeVariable = valeur;

ex : let fruit = “pomme”; ⇒ chaînes de caractères entre double quote
let age = 25; ⇒ nombres, sans double quote

Les noms de variables doivent être en anglais

Modification :

nomDeVariable = valeur;

il “suffit” d’enlever le let qui ne sert qu’à la déclaration.

2. CONSTANTES

Déclaration :

const nomDeVariable = valeur;

ex: const fruit = “pomme”; ⇒ chaînes de caractères entre double quote
const age = 25; ⇒ nombres, sans double quote

Modification :

Les constantes ne sont pas modifiables du tout, d’ailleurs on finira par les utiliser pour intégrer nos fonctions “en dur”

Appeler une variable ou une constante dans une chaine de caractère :

```
let fruit = framboise;
console.log(`Mon fruit préféré c’est : ${fruit}`);
```

Backquote obligatoire pour encadrer la string sinon pas d’affichage de variable

Utilisation du prompt pour déclarer une variable :

let nomVariable = prompt(“Question à l’utilisateur”);

3. FONCTIONS

Création :

```
function nomFonction () {
    instruction;
}
```

Appeler la fonction :

nomFonction (argument)

Fonctionnement :

Les instructions de la fonction s’appliquent à l’argument

ex :

déclaration de la fonction en pseudocode et en code

```
function fonctionQuiMultiplieParQuatre (x) {
    Multiplier l’argument par 4 et afficher le résultat;
}

function fonctionQuiMultiplieParQuatre (x) {
    x*4
    console.log(x*4);
}
```

Appel de la fonction en pseudocode et en code

fonctionQuiMultiplieParQuatre (5) ⇒ résultat : 20

fonctionQuiMultiplieParQuatre (5) ⇒ résultat : 20

Une variable qui est déclarée HORS d'une fonction peut s'utiliser dans une fonction ET hors d'une fonction
Une variable qui est déclarée DANS une fonction ne peut PAS s'utiliser hors de ladite fonction

Les instructions de la fonction s'appliquent à l'argument

les fonctions qui interagissent avec l'utilisateur (par exemple, il se passe qqchose uniquement si l'util. clique sur qqchose) = Event Listener

OBJETS

1. TABLEAUX

array = tableau qui sera un object en js
let yyy = [x, x, x] avec les crochets on crée une liste d'info = tableau qui nous servira à gérer des données

ajouter un élément au tableau :
nomDuTableau.push(truc à ajouter);
nomDuTableau.splice(index de l'élément à partir duquel on supprime, nombre d'éléments à supprimer);
nomDuTableau.splice(1) ⇒ supprime tous les éléments sauf le 1er
nomDuTableau.length = 1 ⇒ supprime tous les éléments sauf le 1er

STRUCTURES CONDITIONNELLES

1. SYNTAXE :

```
if (Condition) {
Instruction;
}
else if (condition) {
instruction;
}
else {
instruction;
}
```

Pour éviter les problématiques de casse :
.toLowerCase
.toUpperCase

2. OPÉRATEURS :

COMPARAISON		LOGIQUE	
==	égal à	&&	ET
===	strictement égal à	||	OU
!=	inégal à	!	NON
!==	strictement inégal à		
>	Strictement supérieur à	++	Incrémenter de 1
>=	Supérieur ou égal à	--	Décrémenter de 1
<	Strictement inférieur à	+	Plus unaire *
<=	Inférieur ou égal à	-	Négation unaire *

MATHÉMATIQUES			
+	Addition (x + y)	*	Multiplication (x * y)
-	Soustraction (x - y)	%	Reste (5 % 2) ⇒ 2
/	Division (x / y)	**	Exponentation (x ** y) ⇒ x^y

* Plus utaire : l'opérateur plus unaire (+) précède son opérande. Converti l'opérande en nombre si possible
*Négation unaire : l'opérateur négation unaire (-) précède son opérande et prend l'opposé de celui-ci (après l'avoir converti en nombre si besoin)

3. SWITCH :

Autre type de structure conditionnelle très adapté à JS qui est un langage de script :

```
switch (expression){
        case valeur 1 :
                Instruction si expression = valeur1;
                break;
        case valeur 2 :
                Instruction si expression = valeur 2;
                break;
        case valeur 3 :
                Instruction si expression = valeur 3;
                break;
        default :
                Instruction si expression != valeurs possibles
}
```

Chaque case est l'équivalent d'un if
le default est l'équivalent d'un else
Le break; est INDISPENSABLE : si pas de break, le programme exécutera toutes les instructions jusqu'à trouver un break;

BOUCLES

1. BOUCLES WHILE

Permet de répéter une opération jusqu'à ce qu'une condition soit atteinte

```
let chiant = prompt("Hey ! on peut aller chez Disney ?? ");
while (chiant.toLowerCase() !="oui") {
    chiant = prompt("Allez, stp, stp, stp !!! On y va ?");
}
console.log("Youpi ! je vais préparer mes bagages !");
```

2. BOUCLES FOR

Répéter une instruction un nombre fini de fois

```
for (let chocolat = 0; chocolat <=10;) {
        chocolat++;
        console.log(chocolat);
}
```

```
for (je déclare la variable chocolat qui contient 0 ; for chocolat qui contient moins que 10;) {
        incrémenter chocolat de 1
        instruction afficher à quoi est égale la variable à chaque tour de boucle;
}
```

Si la variable chocolat doit être ré-utilisée dans le code, il faut la déclarer avant la boucle :

```
let chocolat = 0;
for (chocolat; chocolat <=10;) {
        chocolat++;
        console.log(chocolat);
}
```

3. BOUCLE FOREACH

Dans le tableau list :

```
let list =["Romain", "Inès", "Capucine", "Sofian", "Nico", "Matthieu", "Karo",
"Anna", "Victor", "Gaston"]
```

On veut afficher chacun des éléments contenus dans le tableau. On utilise (Attention syntaxe mauvaise) :

```
list.forEach(function(apprenant) {
        console.log(apprenant);
});
```

⇒ Pour chaque élément contenu dans le tableau, on applique une fonction “fantôme” qui nomme chaque élément du tableau “apprenant” et les affiche (instruction console.log)

La bonne syntaxe est la suivante :

```
list.forEach(apprenant => {
    console.log(apprenant);
});
```

LIER HTML-CSS ET JS

Dans la console :
document.getElementById(xxx) ⇒ Permet “d’attraper” les éléments avec l’id xxx
document.getElementsByClassName(yyy)[z] ⇒ Permet “d’attraper” les éléments avec la classe voulue et l’index z dans la liste des éléments qui ont la classe y
Pour cibler des éléments qui ont déjà des id ou des class qui sont utiles pour le CSS. Interdit de créer des id ou des class uniquement pour JS

document.querySelector(aaa) ⇒ Permet “d’attraper” le premier élément aaa avec :
#aaa ⇒ id aaa
.aaa ⇒ class aaa
“p” ⇒ <p>
...

document.querySelectorAll(aaa) ⇒ Permet “d’attraper” tous les éléments aaa avec :
.aaa ⇒ class aaa
“p” ⇒ <p>
...
document.querySelectorAll(aaa)[z] ⇒ Permet “d’attraper” l’éléments aaa avec l’index [z]

En tapant document.querySelectorAll(aaa)[z] dans l’IDE, ça indique l’élément auquel on va appliquer du code JavaScript. on peut le mettre dans une variable pour en simplifier la manipulation :
let nomVariable = document.querySelectorAll(aaa)[z]
ainsi, en utilisant nomVariable, on manipulera l’élément que l’on a sélectionné.

document.querySelectorAll(aaa)[z].classList ⇒ donne la liste des class attribuées à un élément

Dans CSS, créer une propriété pour une nouvelle class ex :

```
.classTest {
    font-weight: bold;
    color: darkslategrey;
}
```

Dans la console :
document.querySelectorAll("li")[3].classList.add("classTest")
⇒ ça ajoute la class classTest au 4ème “li” du fichier html

Dans la console :
document.querySelectorAll("li")[3].classList.remove("classTest")
⇒ ça enlève la class classTest au 4ème “li” du fichier html

Ordre d’importance :
ul, li, #id
ul, li, class
#id
.class
ul, li
li

donc un élément qui a déjà un id ne sera pas impacté par un ajout de class sur ce que contient déjà l’id
ex : <li id=”ploum”>test
#ploum ⇒ color : blue ; .tralala ⇒ color : red; font-size : 50px;
en appliquant .tralala la taille changera mais pas la couleur
Par contre, pour forcer le changement, après la valeur, taper !important sur la propriété qui sera bloquée (ici la couleur)

Modifier le style directement depuis la console :
document.querySelectorAll("li")[3].style.color = "red"

```
document.querySelector(".title").addEventListener("click", function() {


});
```

“click” = type d'événement, ça peut être hoover, double-click... cf liste d’events dans ressources

Tout à fait possible de créer un event listener qui agit sur un/des élément/s autre/s que celui sur lequel on a cliqué

```
bigTitle.addEventListener("click", function() {
    li4.classList.add("modifiedTitle");
});
```

Si on veut une action annulable en re-clickant : “toggle” au lieu de “add”

```
bigTitle.addEventListener("click", function() {
    bigTitle.classList.toggle("modifiedTitle");
});
```

ATTENTION ! avec l’event “mouseover” si l’on souhaite un retour à la normale dès que la souris quitte l’élément survolé, il faut créer un event “mouseout” avec un toggle :

```
let lorem = document.querySelector("#lorem");
```

```
lorem.addEventListener("mouseover", function() {
    lorem.classList.toggle("modifiedTitle");
});

lorem.addEventListener("mouseout", function() {
    lorem.classList.toggle("modifiedTitle");
});
```

Syntaxe plus correcte :

```
lorem.addEventListener("mouseout", () => {
    lorem.classList.toggle("modifiedTitle");
});
```

Pour récupérer une valeur d'un autre élément

```
document.querySelector("button").addEventListener("click", () => {
    console.log(document.querySelector("input").value);
});
```

Injection code (attention dangereux)

```
let dot = document.querySelectorAll(".listElements")[2]


dot.innerHTML = "treize"
```

RESSOURCES

Cours Anthony :
https://www.codepile.net/pile/dwm12js

Cours OpenClassrooms :
https://openclassrooms.com/fr/courses/2984401-apprenez-a-coder-avec-javascript

Opérateurs (developer mozilla) :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/Op%C3%A9rateurs_arithm%C3%A9tiques

Liste events :
https://developer.mozilla.org/fr/docs/Web/Events

---

let test="bonjour";
test.toUppercase; ⇒ permet au programme de comprendre la string bonjour comme étant en capitales

Document Object Model = DOM = Contenant de la page web. Depuis le DOM on peut travailler JS qui va impacter le html

Ordre d’importance :

ul, li, #id
ul, li, class
#id
.class
ul, li
li