Partie 1 - Les bases du Javascript
Introduction au Javascript

Qu'est-ce que le Javascript ?
Qu'est-ce que c'est ?

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

Un langage de programmation

Tout d'abord, un **langage de programmation** est un langage qui permet aux développeurs d'écrire du **code source** qui sera analysé par l'ordinateur.

Le code source est écrit par le développeur. C'est un ensemble d'actions, appelées instructions, qui vont permettre de donner des ordres à l'ordinateur afin de faire fonctionner le programme.

En fonction du code source, l'ordinateur exécute différentes actions, comme ouvrir un menu, démarrer une application, effectuer une recherche, enfin bref, tout ce que l'ordinateur est capable de faire.

Programmer des scripts

Le Javascript permet de programmer des scripts.

Les scripts sont majoritairement interprétés.

Il est donc nécessaire de posséder un interpréteur pour faire fonctionner du code Javascript, et un interpréteur, vous en utilisez un fréquemment : il est inclus dans votre navigateur Web !

Langage orienté objet

Un langage de programmation orienté objet est un langage qui contient des éléments, appelés objets, et que ces différents objets possèdent des caractéristiques spécifiques ainsi que des manières différentes de les utiliser.

Le Javascript, le langage de scripts

Le Javascript s'inclut directement dans la page Web (ou dans un fichier externe) et permet de dynamiser une page HTML, en ajoutant des interactions avec l'utilisateur, des animations, de l'aide à la navigation, comme par exemple :

- Afficher/masquer du texte ;
- Faire défiler des images ;
- Créer un diaporama avec un aperçu « en grand » des images ;
- Créer des infobulles.

Le Javascript est un langage dit client-side, c'est-à-dire que les scripts sont exécutés par le navigateur chez l'internaute (le client).

Premiers pas en Javascript
Afficher une boîte de dialogue
Le Hello World!

```
<!DOCTYPE html>
<html>
	<head>
		<title>Hello World!</title>
	</head>
	<body>
		<script>
			alert('Hello world!');
		</script>
	</body>
</html>
```

Les nouveautés

Code : JavaScript
```
alert('Hello world!');
```

Il s'agit d'une instruction, c'est-à-dire une commande, un ordre, ou plutôt une action que l'ordinateur va devoir réaliser.

Dans cet exemple, il n'y a qu'une instruction : l'appel de la fonction alert().

La syntaxe du Javascript
Les instructions

Code : JavaScript
```
instruction_1;
instruction_2;
instruction_3;
```

La compression des scripts

Si vous avez oublié un seul point-virgule, votre code compressé ne fonctionnera plus, puisque les instructions ne seront pas correctement séparées.

Les espaces

Le Javascript n'est pas sensible aux espaces.

Les commentaires

Les commentaires sont des annotations faites par le développeur pour expliquer le fonctionnement d'un script, d'une instruction.

Commentaires de fin de ligne
Code : JavaScript

```
instruction_1; // Ceci est ma première instruction
instruction_2;
// La troisième instruction ci-dessous :
instruction_3;
```

Commentaires multilignes
Code : JavaScript

```
/* Ce script comporte 3 instructions : - Instruction 1 qui fait telle chose
- Instruction 2 qui fait autre chose
- Instruction 3 qui termine le script */
instruction_1;
instruction_2;
instruction_3; // Fin du script
```

Les fonctions

Code : JavaScript

```
myFunction(); // « function » veut dire « fonction » en anglais
```

Code : JavaScript

```
alert('Hello world!');
```

Le Javascript « dans la page »

Code : HTML

```
<!DOCTYPE html>
<html>
	<head>
		<title>Hello World!</title>
	</head>
	<body>
		<script>
			alert('Hello world!');
		</script>
	</body>
</html>
```

Le Javascript externe

Code : JavaScript - Contenu du fichier hello.js

```
alert('Hello world!');
```

Code : HTML - Page Web

```
<!DOCTYPE html>
<html>
	<head>
		<title>Hello World!</title>
	</head>
	<body>
		<script src="hello.js"></script>
	</body>
</html>
```

En résumé

- Les instructions doivent êtres séparées par un point-virgule.
- Un code Javascript bien présenté est plus lisible et plus facilement modifiable.
- Il est possible d'inclure des commentaires au moyen des caractères //, /* et */.
- Les codes Javascript se placent dans un élément <script>.
- Il est possible d'inclure un fichier Javascript grâce à l'attribut src de l'élément <script>.

Les variables

Qu'est-ce qu'une variable ?

Pour faire simple, une variable est un espace de stockage sur votre ordinateur permettant d'enregistrer tout type de donnée, que ce soit une chaîne de caractères, une valeur numérique ou bien des structures un peu plus particulières.

Déclarer une variable

Pour déclarer une variable, il vous suffit d'écrire la ligne suivante :

Code : JavaScript

```
let myVariable;
```

Le mot-clé let est présent pour indiquer que vous déclarez une variable.

Code : JavaScript

```
let myVariable;
myVariable = 2;
```

Le signe = sert à attribuer une valeur à la variable ; ici nous lui avons attribué le nombre 2.

Les types de variables

Contrairement à de nombreux langages, le Javascript est un langage typé dynamiquement. Cela veut dire, généralement, que toute déclaration de variable se fait avec le mot-clé let sans distinction du contenu, tandis que dans d'autres langages, comme le C, il est nécessaire de préciser quel type de contenu la variable va devoir contenir.

Les opérateurs arithmétiques

Ils sont à la base de tout calcul et sont au nombre de cinq.

Opérateur	Signe
addition	+
soustraction	-
multiplication	*
division	/
modulo	%

Concernant le dernier opérateur, le modulo est tout simplement le reste d'une division. Par exemple, si vous divisez 5 par 2 alors il vous reste 1 ; c'est le modulo !

Quelques calculs simples

Code : JavaScript

```
let result = 3 + 2;
alert(result); // Affiche : « 5 »
```

Code : JavaScript

```
let number1 = 3, number2 = 2, result;
result = number1 * number2;
alert(result); // Affiche : « 6 »
```

Simplifier encore plus vos calculs

Code : JavaScript

```
let number = 3;
number = number + 5;
alert(number); // Affiche : « 8 »
```

Code : JavaScript

```
let number = 3;
number += 5;
alert(number); // Affiche : « 8 »
```

À noter que ceci ne s'applique pas uniquement aux additions mais fonctionne avec tous les autres opérateurs arithmétiques :

- +=
- -=
- *=
- /=
- %=

Initiation à la concaténation et à la conversion des types

L'opérateur + ; en plus de faire des additions, il permet de faire ce que l'on appelle des concaténations entre des chaînes de caractères.

La concaténation

Code : JavaScript

```
let hi = 'Bonjour', name = 'toi', result;
result = hi + name;
alert(result); // Affiche : « Bonjourtoi »
```

Code : JavaScript

```
let text = 'Bonjour ';
text += 'toi';
alert(text); // Affiche « Bonjour toi ».
```

Interagir avec l'utilisateur

Code : JavaScript

```
let userName = prompt('Entrez votre prénom :');
alert(userName); // Affiche le prénom entré par l'utilisateur : Patrick
```

Code : JavaScript

```
let start = 'Bonjour ', name, end = ' !', result;
name = prompt('Quel est votre prénom ?');
result = start + name + end;
alert(result);// Affiche Bonjour Patrick !
```

Convertir une chaîne de caractères en nombre

Code : JavaScript

```
let text1 = '1337', number1;
number1 = parseInt(text1);
alert(typeof number1); // Affiche : « number »
alert(number1); // Affiche : « 1337 »
```

En résumé

- Une variable est un moyen pour stocker une valeur.
- On utilise le mot clé let pour déclarer une variable, et on utilise = pour affecter une valeur à la variable.
- Les variables sont typées dynamiquement, ce qui veut dire que l'on n'a pas besoin de spécifier le type de contenu que la variable va contenir.
- Grâce à différents opérateurs, on peut faire des opérations entre les variables.
- L'opérateur + permet de concaténer des chaînes de caractères, c'est-à-dire de les mettre bout à bout.
- La fonction prompt() permet d'interagir avec l'utilisateur.

Les conditions

La base de toute condition : les booléens

À quoi vont-ils nous servir ? À obtenir un résultat comme true (vrai) ou false (faux) lors du test d'une condition.

Les opérateurs de comparaison

Comme leur nom l'indique, ces opérateurs vont permettre de comparer diverses valeurs entre elles. En tout, ils sont au nombre de huit, les voici :

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

Code : JavaScript

```
let number = 4, text = '4', result;
result = number == text;
alert(result); // Affiche « true » alors que « number » est un nombre et « text » une chaîne de caractères
result = number === text;
alert(result); // Affiche « false » car cet opérateur compare aussi les types des variables en plus de leurs valeurs
```

Les opérateurs logiques

Ils sont au nombre de trois :

Opérateur	Type de logique	Utilisation
&&	**ET**	**valeur1 && valeur2**
||	**OU**	**valeur1 || valeur2**
!	**NON**	**!valeur**

L'opérateur ET

Cet opérateur vérifie la condition lorsque toutes les valeurs qui lui sont passées valent true. Si une seule d'entre elles vaut false alors la condition ne sera pas vérifiée. Exemple :

Code : JavaScript

```
let result = true && true;
alert(result); // Affiche : « true »

result = true && false;
alert(result); // Affiche : « false »

result = false && false;
alert(result); // Affiche : « false »
```

L'opérateur OU

Cet opérateur est plus « souple » car il renvoie true si une des valeurs qui lui est soumise contient true, qu'importent les autres valeurs. Exemple :

Code : JavaScript

```
let result = true || true;
alert(result); // Affiche : « true »

result = true || false;
alert(result); // Affiche : « true »

result = false || false;
alert(result); // Affiche : « false »
```

L'opérateur NON

Cet opérateur se différencie des deux autres car il ne prend qu'une seule valeur à la fois. S'il se nomme « NON » c'est parce que sa fonction est d'inverser la valeur qui lui est passée, ainsi true deviendra false et inversement. Exemple :

```
let result = false;
result = !result; // On stocke dans « result » l'inverse de « result », c'est parfaitement possible
alert(result); // Affiche « true » car on voulait l'inverse de « false »

result = !result;
alert(result); // Affiche « false » car on a inversé de nouveau « result », on est donc passé de « true » à « false »
```

Combiner les opérateurs

Code : JavaScript
```
let condition1, condition2, result;
condition1 = 2 > 8; // false
condition2 = 8 > 2; // true

result = condition1 && condition2;
alert(result); // Affiche « false »
```

Code : JavaScript
```
let result = 2 > 8 && 8 > 2;
alert(result); // Affiche « false »
```

La condition « if else »

La structure if pour dire « si »

Code : JavaScript
```
if (true) {
	alert("Ce message s'est bien affiché.");
}
if (false) {
	alert("Pas la peine d'insister, ce message ne s'affichera pas.");
}
```

La structure else pour dire « sinon »

Code : JavaScript
```
if ( /* condition */ ) { // Du code...
} else {
// Du code...
}
```

La structure else if pour dire « sinon si »

Code : JavaScript
```
let floor = parseInt(prompt("Entrez l'étage où l'ascenseur doit se rendre (de -2 à 30) :"));
```

```
if (floor == 0) {
	alert("Vous vous trouvez déjà au rez-de-chaussée.");
} else if (-2 <= floor && floor <= 30) {
	alert("Direction l'étage n°" + floor + "!");
} else {
alert("L'étage spécifié n'existe pas.");
}
```

La condition « switch »

Code : JavaScript

```
let drawer = parseInt(prompt("Choisissez le tiroir à ouvrir (1 à 4) :"));
switch (drawer) {
	case 1:
		alert("Contient divers outils pour dessiner : du papier, des crayons, etc.");
		break;
	case 2:
		alert("Contient du matériel informatique : des câbles, des composants, etc.");
		break;
	case 3:
		alert("Ah ? Ce tiroir est fermé à clé ! Dommage !");
		break;
	case 4:
		alert("Contient des vêtements : des chemises, des pantalons, etc.");
		break;
	default:
		alert("Info du jour : le meuble ne contient que 4 tiroirs et, jusqu'à preuve du contraire, les tiroirs négatifs n'existent pas.");
}
```

- On écrit le mot-clé switch suivi de la variable à analyser entre parenthèses et d'une paire d'accolades ;
- Dans les accolades se trouvent tous les cas de figure pour notre variable, définis par le mot-clé case suivi de la valeur qu'il doit prendre en compte (cela peut être un nombre mais aussi du texte) et de deux points ;
- Tout ce qui suit les deux points d'un case sera exécuté si la variable analysée par le switch contient la valeur du case ;
- À chaque fin d'un case on écrit l'instruction break pour « casser » le switch et ainsi éviter d'exécuter le reste du code qu'il contient ;
- Et enfin on écrit le mot-clé default suivi de deux points. Le code qui suit cette instruction sera exécuté si aucun des cas précédents n'a été exécuté. Attention, cette partie est optionnelle, vous n'êtes pas obligés de l'intégrer à votre code.

En résumé

- Une condition retourne une valeur booléenne : true ou false.
- De nombreuxopérateurs existent afin de tester des conditions et ils peuvent être combinés entre eux.
- La condition if else est la plus utilisée et permet de combiner les conditions.
- Quand il s'agit de tester une égalité entre une multitude de valeurs, la condition switch est préférable.

Les boucles

L'incrémentation

Code : JavaScript

```
let number = 0;
number = number + 1;
```

Le fonctionnement

Code : JavaScript

```
let number = 0;
number++;
alert(number); // Affiche : « 1 »

number--;
alert(number); // Affiche : « 0 »
```

L'ordre des opérateurs

Code : JavaScript

```
let number_1 = 0;
let number_2 = 0;
number_1++;
++number_2;
alert(number_1); // Affiche : « 1 »
alert(number_2); // Affiche : « 1 »
```

Code : JavaScript

```
let number = 0;
let output = ++number;
alert(number); // Affiche : « 1 »
alert(output); // Affiche : « 1 »
```

Maintenant, si on place l'opérateur après la variable à incrémenter, l'opération retourne la valeur de number avant qu'elle ne soit incrémentée :

Code : JavaScript

```
let number = 0;
let output = number++;
alert(number); // Affiche : « 1 »
alert(output); // Affiche : « 0 »
```

La boucle while

Répéter tant que...

Code : JavaScript

```
while (condition) {
instruction_1;
instruction_2;
instruction_3;
}

let number = 1;
while (number < 10) {
number++;
}
alert(number); // Affiche : « 10 »
```

La boucle do while

Code : JavaScript

```
do { instruction_1;
instruction_2;
instruction_3;
} while (condition);
```

La boucle for

Code : JavaScript

```
for (initialisation; condition; incrémentation) {
instruction_1;
instruction_2;
instruction_3;
}
```

Code : JavaScript

```
for (let iter = 0; iter < 5; iter++) {
alert("Itération n°" + iter);
}
```

En résumé

- L'incrémentation est importante au sein des boucles. Incrémenter ou décrémenter signifie ajouter ou soustraire une unité à une variable. Le comportement d'un opérateur d'incrémentation est différent s'il se place avant ou après la variable.
- La boucle while permet de répéter une liste d'instructions tant que la condition est vérifiée.
- La boucle do while est une variante de while qui sera exécutée au moins une fois, peu importe la condition.
- La boucle for est une boucle utilisée pour répéter une liste d'instructions un certain nombre de fois. C'est donc une variante très ciblée de la boucle while.

Les fonctions

Concevoir des fonctions

Dans les chapitres précédents vous avez découvert quatre fonctions : alert(), prompt(), confirm() et parseInt().

Créer sa première fonction

Code : JavaScript

```
function myFunction(arguments) {
// Le code que la fonction va devoir exécuter
}
```

- Le mot-clé function est présent à chaque déclaration de fonction. C'est lui qui permet de dire « Voilà, j'écris ici une fonction ! » ;
- Vient ensuite le nomde votre fonction, ici myFunction;
- S'ensuit un couple de parenthèses contenant ce que l'on appelle des arguments. Ces arguments servent à fournir des informations à la fonction lors de son exécution. Par exemple, avec la fonction alert() quand vous lui passez en paramètre ce que vous voulez afficher à l'écran ;
- Et vient enfin un couple d'accolades contenant le code que votre fonction devra exécuter.

Code : JavaScript

```
function showMsg() {
alert("Et une première fonction, une !");
}

showMsg(); // On exécute ici le code contenu dans la fonction
```

La portée des variables

Code : JavaScript

```
let ohai = "Hello world !";
function sayHello() {
        alert(ohai);
}

sayHello();
```

Les variables globales

À l'inverse des variables locales, celles déclarées en-dehors d'une fonction sont nommées les variables globales car elles sont accessibles partout dans votre code, y compris à l'intérieur de vos fonctions.

Code : JavaScript

```
let var1 = 2, var2 = 3; // Ici les variables sont globales
function calculate() {
	alert(var1 * var2); // Affiche : « 6 »
}
calculate();
```

Par principe, cette façon de faire est stupide : vu que ces variables ne servent qu'à la fonction calculate(), autant les déclarer dans la fonction de la manière suivante :

Code : JavaScript

```
function calculate() {
	let var1 = 2, var2 = 3; // Ici les variables sont locales
	alert(var1 * var2);
}
calculate();
```

Les arguments

Créer et utiliser un argument

Code : JavaScript

```
function myFunction(arg) { // Notre argument est la variable « arg »
	// Une fois que l'argument a été passé à la fonction, vous allez le retrouver dans la variable « arg »
	alert("Votre argument : " + arg);
}
myFunction("En voilà un beau test !");
```

La portée des arguments

Code : JavaScript

```
function scope(arg) {
	// Au début de la fonction, la variable « arg » est créée avec le contenu de l'argument qui a été passé à la fonction
	alert(arg); // Nous pouvons maintenant utiliser l'argument comme souhaité : l'afficher, le modifier, etc.
	// Une fois l'exécution de la fonction terminée, toutes les variables contenant les arguments sont détruites
}
```

Les arguments multiples

Code : JavaScript

```
function moar(first, second) {
        // On peut maintenant utiliser les variables « first » et «second » comme on le souhaite :
        alert("Votre premier argument : " + first); // Affiche 'Un !'
        alert("Votre deuxième argument : " + second); // Affiche 'Deux!'
}

moar("Un !", "Deux !");
```

Les valeurs de retour

Code : JavaScript

```
function sayHello() {
        return "Bonjour !";
        alert("Attention ! Le texte arrive !");
}
alert(sayHello());
```

Comme vous pouvez le constater, notre premier alert() ne s'est pas affiché !
Cela s'explique par la présence du return : cette instruction met fin à la fonction, puis retourne la valeur.

Les fonctions anonymes

Comme leur nom l'indique, ces fonctions spéciales sont anonymes car elles ne possèdent pas de nom !

Code : JavaScript

```
let sayHello = (function() {
        return "Yop !";
})();
alert(sayHello); // Affiche : « Yop ! »
```

Explication : notre fonction a été appelée par le biais du nom de la variable à laquelle nous l'avons affectée.

En résumé

- Il existe des fonctions natives, mais il est aussi possible d'en créer, avec le mot-clé **function**.
- Les variables déclarées avec **var** au sein d'une fonction ne sont accessibles que dans cette fonction.
- Il faut éviter le plus possible d'avoir recours aux variables globales.
- Une fonction peut recevoir un nombre défini ou indéfini de paramètres. Elle peut aussi retourner une valeur ou ne rien retourner du tout.

Les objets et les tableaux

Introduction aux objets

Le Javascript est un langage orienté objet. Cela veut dire que le langage dispose d'objets.

Que contiennent les objets ?

Les objets contiennent trois choses distinctes :

- Un constructeur ;
- Des propriétés ;
- Des méthodes.

Le constructeur

Le constructeur est un code qui est exécuté quand on utilise un nouvel objet. Il permet d'effectuer des actions comme définir diverses variables au sein même de l'objet (comme le nombre de caractères d'une chaîne de caractères). Tout cela est fait automatiquement pour les objets natifs, nous en reparlerons quand nous aborderons l'orienté objet.

Les propriétés

Toute valeur va être placée dans une variable au sein de l'objet : c'est ce que l'on appelle une propriété. Une propriété est une variable contenue dans l'objet, elle contient des informations nécessaires au fonctionnement de l'objet.

Les méthodes

Enfin, il est possible de modifier l'objet. Cela se fait par l'intermédiaire des méthodes. Les méthodes sont des fonctions contenues dans l'objet, et qui permettent de réaliser des opérations sur le contenu de l'objet. Par exemple, dans le cas d'une chaîne de caractères, il existe une méthode qui permet de mettre la chaîne de caractères en majuscules.

Les tableaux

Code : JavaScript

```
let myArray = ['Sébastien', 'Laurence', 'Ludovic', 'Pauline', 'Guillaume'];
```

On peut schématiser le contenu du tableau myArray ainsi :

Indice	0	1	2	3	4
Donnée	Sébastien	Laurence	Ludovic	Pauline	Guillaume

Récupérer et modifier des valeurs

Code : JavaScript

```
let myArray = ['Sébastien', 'Laurence', 'Ludovic', 'Pauline', 'Guillaume'];
alert(myArray[1]); // Affiche : « Laurence »
```

Code : JavaScript

```
let myArray = ['Sébastien', 'Laurence', 'Ludovic', 'Pauline', 'Guillaume'];
myArray[1] = 'Clarisse';
alert(myArray[1]); // Affiche : « Clarisse »
```

Opérations sur les tableaux

Ajouter et supprimer des items

Code : JavaScript

```
var myArray = ['Sébastien', 'Laurence', 'Ludovic', 'Pauline', 'Guillaume'];
myArray.shift(); // Retire « Sébastien »
myArray.pop(); // Retire « Guillaume »

alert(myArray); // Affiche « Laurence,Ludovic,Pauline »
```

Chaînes de caractères et tableaux

Les chaînes de caractères possèdent une méthode split() qui permet de les découper en un tableau, en fonction d'un séparateur. Prenons l'exemple suivant :

Code : JavaScript

```
let cousinsString = 'Pauline Guillaume Clarisse',
    cousinsArray = cousinsString.split(' ');
alert(cousinsString);
alert(cousinsArray);
```

L'inverse de split(), c'est-à-dire créer une chaîne de caractères depuis un tableau, se nomme join() :

Code : JavaScript

```
let cousinsString_2 = cousinsArray.join(' ');
alert(cousinsString_2);
```

Parcourir un tableau
Parcourir avec for

Code : JavaScript
```
let myArray = ['Sébastien', 'Laurence', 'Ludovic', 'Pauline', 'Guillaume'];
```

Code : JavaScript
```
for (let i = 0, c = myArray.length; i < c; i++) {
        alert(myArray[i]);
}
```

On commence par définir la variable de boucle i.
On définit une seconde variable, dans le bloc d'initialisation, qui contiendra la valeur de length.
myArray.length représente le nombre d'items du tableau, c'est-à-dire cinq.

Ensuite, on règle la condition pour que la boucle s'exécute tant que l'on n'a pas atteint le nombre d'items.

Les objets littéraux
Code : JavaScript
```
let family = {
        self:       'Sébastien',
        sister:     'Laurence',
        brother:    'Ludovic',
        cousin_1:   'Pauline',
        cousin_2:   'Guillaume'
};
```

La syntaxe d'un objet
Code : JavaScript
```
let myArray = []; // Créer un array vide
```

Code : JavaScript
```
let myObject = {}; // Créer un objet vide
```

Code : JavaScript

```
let myObject = {
        item1 : 'Texte 1',
        item2 : 'Texte 2'
};
```

Accès aux items

Code : JavaScript

```
family.sister;
```

ou

Code : JavaScript

```
family['sister'];
```

Code : JavaScript

```
let id = 'sister';
alert(family[id]); // Affiche : « Laurence »
```

Utilisation des objets littéraux

Code : JavaScript

```
function getCoords() {
/* Script incomplet, juste pour l'exemple */
return { x: 12, y: 21 };
}
let coords = getCoords();
alert(coords.x); // 12
alert(coords.y); // 21
```

La valeur de retour de la fonction getCoords() est mise dans la variable coords, et l'accès à x et y en est simplifié.

En résumé

- Un objet contient un constructeur, des propriétés et des méthodes.
- Les tableaux sont des variables qui contiennent plusieurs valeurs, chacune étant accessible au moyen d'un indice.
- Les indices d'un tableau sont toujours numérotés à partir de 0. Ainsi, la première valeur porte l'indice 0.
- Des opérations peuvent être réalisées sur les tableaux, comme ajouter des items ou en supprimer.
- Pour parcourir un tableau, on utilise généralement une boucle for, puisqu'on connaît, grâce à la propriété length, le nombre d'items du tableau.
- Les objets littérauxsont une variante des tableauxoù chaque itemest accessible via un identifiant et non un indice.