

JavaScript

JS





Qu'est-ce que le JavaScript ?

JavaScript est un langage de programmation qui permet de créer des pages web dynamiques.

Il est principalement utilisé pour :

- 📌 Créer des animations
- 📌 Créer des jeux
- 📌 Créer des applications web
- 📌 Créer des applications mobiles



Avantages du Javascript

- 📌 **Le JavaScript est un langage de programmation léger. Cela signifie que le code JavaScript est plus facile à lire et à écrire. De plus, le JavaScript est un langage de programmation qui est facile à apprendre et à utiliser.**



Avantages du Javascript

- 📌 Le JavaScript est un langage de programmation multiplateforme. Cela signifie que le code JavaScript peut être utilisé sur plusieurs plates-formes. Le JavaScript peut être utilisé sur les navigateurs Web, les serveurs Web, les appareils mobiles, les ordinateurs de bureau, etc.



Avantages du Javascript

- 📌 **Le JavaScript est un langage de programmation orienté objet. Cela signifie que le JavaScript peut être utilisé pour créer des objets. Les objets sont des entités qui peuvent être utilisées pour représenter des choses réelles. Dans l'orienté objet, nous codons en nous posant les questions :
"Quelle est la chose que je veux représenter ?" et "Quelles sont les caractéristiques de cette chose ?"**



Avantages du Javascript

- 📌 Le JavaScript est un langage de programmation dynamique. Cela signifie que le JavaScript peut être utilisé pour créer des pages Web dynamiques. Les pages Web dynamiques sont des pages Web qui peuvent être modifiées en fonction de l'utilisateur, du contexte, etc.



Avantages du JavaScript

- 📌 Le JavaScript est un langage de programmation interprété. Cela signifie que le JavaScript peut être exécuté directement par un interpréteur sans avoir besoin d'être compilé.



Avantages du JavaScript

- 📌 Le JavaScript est un langage de programmation multi-paradigme. Cela signifie que le JavaScript peut être utilisé dans plusieurs paradigmes de programmation. Le JavaScript peut être utilisé dans les paradigmes de programmation orientée objet, fonctionnelle, impératifs et déclaratifs.



Inconvénients du Javascript

- 📌 Le JavaScript est un langage de programmation non typé. Cela signifie que le JavaScript n'a pas besoin d'être typé. Dans les langages de programmation typés, les variables doivent être déclarées avec un type de données spécifique. En JavaScript, les variables peuvent être déclarées sans type de données spécifique.



Inconvénients du Javascript

- 📌 Le JavaScript est un langage de programmation lent. Cela signifie que le JavaScript est plus lent que d'autres langages de programmation tels que C ou C++.



La portée de variables en JavaScript

La portée ou le scope de variables est la zone de code où une variable est accessible. Les variables peuvent avoir une portée globale ou locale. Une variable globale est accessible dans tout le code. Une variable locale est accessible uniquement dans la fonction dans laquelle elle est définie.



La portée de variables en JavaScript

Exemple:

```
let a = 5; // La variable a est globale

function multiplier() {
    let b = 6; // La variable b est locale à la fonction multiplier
    return a * b;
}
console.log(multiplier()); // 30
console.log(b); // ReferenceError: b is not defined
```



La portée de bloc en JavaScript

La portée de bloc est la zone de code où une variable est accessible. Une variable déclarée avec le mot-clé `let` (ou `const`) a une portée de bloc. Une variable déclarée avec le mot-clé `var` a une portée de fonction.



La portée de bloc en JavaScript

Exemple:

```
let a = 5; // La variable a est globale

if (true) {
    let b = 6; // La variable b est locale au bloc if
    console.log(a); // 5
    console.log(b); // 6
}
console.log(a); // 5
console.log(b); // ReferenceError: b is not defined
```



La portée des données

Les fonctions délimitent également leur propre scope. Les variables déclarées dans une fonction ne sont pas accessibles depuis l'extérieur de celle-ci à moins d'être retournées, qu'elles aient été déclarées avec var, let ou const.



La portée des données

Les variables globales, elles, sont déclarées en dehors d'une fonction ou d'un bloc et sont accessibles depuis n'importe quelle partie du fichier.



Les fonctions en Javascript



Les fonctions en JavaScript

Une fonction est une suite d'instructions.

Aux fonctions peuvent être passées des paramètres, qui peuvent être n'importe quel objet, valeur ou fonction.

Une fonction peut retourner une valeur.



Les fonctions en JavaScript

Pour déclarer une fonction, il nous faut utiliser le mot-clé **function** suivi :

- 📌 **Du nom de la fonction**
- 📌 **D'une liste de paramètres , entre parenthèses et séparés par des virgules**
- 📌 **Puis le body, les instructions de la fonction définies entre accolades.**



Les fonctions en JavaScript

Exemple de déclaration de fonction:

```
function add(a, b) { // ici on déclare la fonction add() qui prend a et b en paramètres  
    return a + b; // ici on retourne la somme des nombres entrés en paramètre  
}
```



Les fonctions en JavaScript

Il est important de se souvenir qu'une fonction en JavaScript est un type spécial d'objet. Il contient les instructions de la fonction.

Nous pouvons en créer une avec le constructeur built-in `Function(funcname, body)`.

On peut aussi, puisque c'est un objet, assigner n'importe quelle fonction à une variable:

```
var body = "return a + b";  
var add = Function(a, b, body);
```



Les fonctions en JavaScript

Et nous pouvons passer cette fonction en paramètre d'une autre grâce à cette expression:

```
function add(a, b) {  
    return a + b;  
}  
function callfunc(func, a, b) { // on déclare une fonction qui prend en paramètre une fonction et deux nombres  
    return func(a, b); // on retourne le résultat de la fonction prise en paramètre  
}  
callfunc(add, 1, 2); // 3
```



Les fonctions anonymes en JavaScript

Une fonction anonyme est une fonction sans nom. Une fonction anonyme est souvent stockée dans une variable.

Pour exécuter une fonction anonyme, on peut :

- 📌 L'appeler directement
- 📌 La faire passer en paramètre
- 📌 L'assigner à une variable



Exemples:

```
// En l'appelant directement  
(function() {  
    console.log("Je suis une fonction anonyme");  
})();
```

```
// En l'assignant à une variable  
let x = function() {  
    console.log("Je suis une fonction anonyme");  
};  
x();
```

```
// En la faisant passer en paramètre  
console.log(function() {  
    console.log("Je suis une fonction anonyme");  
});
```




Les fonctions fléchées en JavaScript

Les fonctions fléchées sont des fonctions anonymes. Elles sont écrites de manière plus concise que les fonctions classiques.

Les fonctions fléchées peuvent être utilisées comme des expressions de fonctions ou comme des fonctions de rappel; fonctions de callback.

Les fonctions fléchées ne créent pas leur propre valeur `this` mais héritent de celle du contexte dans lequel elles sont créées.



Les fonctions fléchées en JavaScript

Exemple:

```
// On écrit une fonction fléchée
let x = (a, b) => a * b;

// On l'utilise
console.log(x(5, 6)); // 30

// On peut aussi écrire une fonction fléchée sur plusieurs lignes
let y = (a, b) => {
    let resultat = a * b;
    return resultat;
};
console.log(y(5, 6)); // 30
```



Les fonctions de rappel en JavaScript

Une fonction de rappel est une fonction passée en paramètre d'une autre fonction. Elle est appelée à l'intérieur de la fonction parente.

Les fonctions de rappel en JavaScript



Exemple:

```
function afficher(nom) {  
    console.log(nom);  
}  
  
function afficherListe(noms, fonction) {  
    for (let i = 0; i < noms.length; i++) {  
        fonction(noms[i]);  
    }  
}  
  
let listeNoms = ["Bob", "Alice", "Joe"];  
afficherListe(listeNoms, afficher); // Bob Alice Joe
```



Les fonctions de rappel en JavaScript

Exemple avec une fonction fléchée:

```
function afficherListe(noms, fonction) {  
    for (let i = 0; i < noms.length; i++) {  
        fonction(noms[i]);  
    }  
}  
  
let listeNoms = ["Bob", "Alice", "Joe"];  
afficherListe(listeNoms, nom => console.log(nom)); // Bob Alice Joe
```



Les fonctions auto-invoquées en JavaScript

Une fonction immédiate est une fonction qui est appelée immédiatement après sa déclaration. Les fonctions auto-invoquées sont souvent utilisées pour éviter les conflits de nommage entre variables et fonctions.



Les fonctions auto-invoquées en JavaScript

Exemple:

```
(function() {  
    let a = 5;  
    let b = 6;  
    console.log(a * b);  
})();
```



Les fonctions anonymes auto-invoquées en JavaScript

Une fonction anonyme auto-invoquée est une fonction anonyme qui est appelée immédiatement après sa déclaration. Les fonctions anonymes auto-invoquées sont souvent utilisées pour créer des modules.



Les fonctions anonymes auto-invoquées en JavaScript

Exemple:

```
(function() {  
    let a = 5;  
    let b = 6;  
    console.log(a * b);  
})();
```



Les fonctions internes en JavaScript

Une fonction interne est une fonction définie à l'intérieur d'une autre fonction. Une fonction interne a accès aux variables de la fonction parente.



Les fonctions internes en JavaScript

Exemple:

```
function multiplier(a, b) {  
    function multiplierPar2(x) {  
        return x * 2;  
    }  
    return multiplierPar2(a) * multiplierPar2(b);  
}  
console.log(multiplier(5, 6)); // 120
```

Exercice



Exercice 1: Créer une fonction qui permet de dire bonjour au bout de 1,5 seconde

Exercice 2: Créer une fonction qui a pour prototype `function aumoins3(array, verifcallback) {}` qui permet de vérifier si un tableau contient au moins 3 éléments qui vérifient la condition de la fonction de callback elle retourne true ou false

Exercice 3: Créer une fonction qui a pour prototype `function filter(array, verifcallback) {}` qui permet de renvoyer un tableau contenant tous les éléments qui vérifient la condition de la fonction de callback



Le typage faible

Javascript est un langage à typage faible. Ce qui veut dire que les types de variables ne sont pas donnés explicitement à la déclaration.

let, var, ou const ne nous donnent pas d'information sur le type de variable déclarée.

Le typage faible

Cela nous impose une plus grande rigueur lors de l'écriture de notre code. Puisque le typage est faible, alors chaque fonction accepte n'importe quel type en paramètre.

Dans un langage comme le C++, les types de variables attendus dans une fonction sont explicites, et leur valeur de retour également. Cela nous donne des informations supplémentaires sur le comportement de la fonction et nous évite des erreurs/bugs.


Ce n'est pas le cas en Javascript, nous sommes plus libres mais il revient donc à nous de savoir quelles variables nous envoyons dans quelles fonctions et s'assurer que nous ne faisons pas d'erreur.








Les types de données en JavaScript

 `string` : **chaîne de caractères**

 `number` : **nombre**

 `boolean` : **booléen**

 `null` : **null**

 `undefined` : **non-défini**

 `object` : **objet**

 `symbol` : **symbole**



```
let a = "hello"; // chaîne de caractères
let b = 5; // nombre
let c = true; // booléen
let d = null; // null
let e; // non-défini
let f = { name: "John" }; // objet
let g = Symbol("id"); // symbole
```




Les types objets



Qu'est-ce qu'un type objet ?

Un type objet est une donnée qui peut être modifiée. Les types objets sont:

-  les tableaux
-  les fonctions
-  les objets
-  les dates



Déclaration d'un type objet

```
const tableau = [1, 2, 3];  
const fonction = function() {};  
const objet = { nom: "Aurora" };  
const date = new Date();
```



Les types objet prédéfinis



Qu'est-ce qu'un type objet prédéfini ?

Un type objet prédéfini est un type objet qui est défini par le langage JavaScript. Les types objets prédéfinis sont:

- 📌 les dates
- 📌 les erreurs
- 📌 les math
- 📌 les JSON



Déclaration d'un type objet prédéfini

```
const date = new Date();  
const erreur = new Error("Oups!");  
const math = Math.random();  
const json = JSON.stringify({ nom: "Aurora" });
```



Le Hoisting



Qu'est-ce que le hoisting ?

Le hoisting est un mécanisme par lequel les déclarations de variables et de fonctions sont déplacées au sommet de leur portée (scope) avant l'exécution du code.

Le hoisting n'affecte que les déclarations et pas les initialisations.

Le hoisting



Donc, les variables déclarées avec var dans Javascript peuvent être "utilisées" avant leur déclaration.

```
// Ce que l'on écrit  
x = 5;  
console.log(x); // 5  
var x;
```

```
// Ce que Javascript comprend  
var x;  
x = 5;  
console.log(x)
```

Ce qui peut prêter à confusion si le code est long et complexe.



Le hoisting

Ce que l'on écrit:

```
function test() {  
  console.log(a); // undefined  
  console.log(foo()); // 2  
  
  var a = 1;  
  function foo() {  
    return 2;  
  }  
}  
  
test();
```



Ce que JavaScript comprend:

```
function test() {  
  function foo() { // La déclaration de fonction est déplacée au sommet de sa portée  
    return 2;  
  }  
  var a; // La déclaration de variable également  
  console.log(a); // undefined  
  console.log(foo()); // 2  
  a = 1;  
}  
  
test();
```



Le hoisting

Ce n'est pas le cas avec `let` ou `const`, c'est une raison pour laquelle il est recommandé de les utiliser.

```
x = 5;  
console.log(x); // ReferenceError  
let x;  
  
y = 5;  
console.log(y); // SyntaxError  
const y;
```



Contexte et variables globales



Contexte et variables globales

En JavaScript, le contexte se limite à un objet.

Par défaut, JavaScript cherche toujours la première valeur assignée à une variable lorsqu'elle est utilisée.



Contexte et variables globales

Il vaut mieux tirer parti des contextes afin de garder un code clair et compartimenté, l'utilisation de variables globales n'est pas recommandée.



Créer des Objets en Javascript

On peut créer un objet de deux façons en JavaScript.
Soit de façon littérale et explicite:

```
var panier = {fruit : pomme}
```




Créer des Objets en Javascript

Soit en utilisant un constructeur avec le mot-clé new:

```
var panier = new Object();  
panier.fruit = "pomme"
```

On utilise ici la fonction constructeur Object() native.



Créer des Objets en Javascript

La méthode **constructor** est une méthode spéciale destinée à gérer l'initialisation d'un objet.

Par défaut, les classes font appel à un constructeur vide lors de la création d'un nouvel objet:

```
constructor() {}
```



Ce constructeur est appelé pour n'importe quelle classe de base dont on initialise l'objet sans avoir spécifié de constructeur:

```
Var = new myClass()
```



Les Constructeurs en Javascript

Voici comment doit être utilisé un constructeur dans une classe:

```
class Panier {  
    constructor() {  
        this.fruit = 'banane'; // On assigne 'banane' à l'attribut fruit de Panier  
    }  
    getFruit() {  
        console.log(this.fruit); // On crée une méthode qui affiche le fruit dans la console  
    }  
}  
  
let panier = new Panier();  
panier.getFruit(); // banane
```



Les Constructeurs en Javascript

Nous pouvons faire passer des paramètres à notre constructeur afin d'y entrer des valeurs lors de l'instanciation:

```
class Panier {  
    constructor(fruitname) { // Ici, notre constructeur attend un paramètre à assigner  
        // à assigner à l'attribut fruit de notre classe Panier lorsqu'elle sera instanciée.  
        this.fruit = fruitname;  
    }  
    getFruit() {  
        console.log(this.fruit);  
    }  
}  
let panier = new Panier('banane'); // Nous entrons 'banane' en paramètre lors de l'instanciation  
panier.getFruit(); // banane
```



Les Constructeurs en JavaScript

```
class Panier {  
    constructor(fruitname = "pomme") { // nous pouvons assigner une valeur  
                                        // par défaut au paramètre  
        this.fruit = fruitname;  
    }  
    getFruit() {  
        console.log(this.fruit);  
    }  
}  
let panier1 = new Panier('banane');  
panier1.getFruit(); // banane  
let panier2 = new Panier();  
panier2.getFruit(); //pomme
```



Le mot-clé `this` dans un objet

En JavaScript, le mot-clé `this` est utilisé pour faire référence à l'objet courant.

Dans un objet, cela nous permet d'accéder à une propriété d'un objet à l'intérieur de celui ci.



Le mot-clé `this` dans un objet

Exemple:

```
let objet = {  
    propriete: 10,  
    fonction: function() {  
        return this.propriete;  
    },  
};  
console.log(objet.fonction()) // affiche 10
```




Exercice 1:

Créer un village qui aura les attributs :

- 📌 nbvillageois | 1
- 📌 nbbois | 100
- 📌 nbpierre | 100
- 📌 nbor | 0
- 📌 Batiments | [HDV]



L'héritage avec Javascript



L'héritage avec Javascript

Pour hériter d'un objet en Javascript, nous utilisons le mot-clé

`extends`



L'héritage avec JavaScript

Dans le prochain slide nous créons une classe `Barquette` qui hérite de la classe `Panier`



```
class Panier {
    constructor() {
        this.fruit = "framboises";
    }
    getFruit() {
        console.log(this.fruit);
    }
}

class Barquette extends Panier {
    constructor() {
        super();
        this.maxq = 5;
    }
}

let barquette = new Barquette();
console.log(barquette); // {fruit: "framboise", maxq: 5}
```



L'héritage avec JavaScript

Le mot-clé `static` définit une propriété ou une méthode statique pour une classe. Les méthodes statiques sont appelées sans instancier leur classe et ne peuvent être appelées que depuis la classe.



Exercice 1:

Créer une classe **Personnage** qui aura les attributs:

 **nom**

 **sante**

 **force**



Exercice 2:

Créer une classe Aventurier qui hérite de Personnage et qui aura l'attribut supplémentaire:

 **xp**



Exercice 3:

Créer une classe Guerrier qui hérite de Personnage et qui aura l'attribut supplémentaire:

 **rage**



Exercice 4:

Créer une classe Paladin qui hérite de Aventurier et qui aura l'attribut supplémentaire:

 **vertu**



Exercice 5:

Créer une classe Mage qui hérite de Aventurier et qui aura l'attribut supplémentaire :

 **mana**

Le mot-clé static



Exemple:

```
class Panier {  
    static nbPaniers = 0;  
    static getNbPaniers() {  
        return Panier.nbPaniers;  
    }  
    constructor() {  
        Panier.nbPaniers++;  
    }  
}  
  
let panier1 = new Panier();  
let panier2 = new Panier();  
console.log(Panier.getNbPaniers()); // 2
```



Les Modules en JavaScript

Un module est un fichier JavaScript qui contient du code. Il peut contenir des variables, des classes, des fonctions, etc. Les modules sont utilisés pour organiser le code en petits morceaux réutilisables.



Les Modules en JavaScript

Le mot-clé `export` est utilisé pour exporter des classes, des fonctions, des constantes, etc. du fichier courant vers un autre fichier.



Les Modules en JavaScript

Le mot-clé `import` est utilisé pour importer des classes, des fonctions, des constantes, etc. d'un autre fichier vers le fichier courant.



Exemple:

```
// Dans le fichier Panier.js
export class Panier {
  constructor() {
    this.fruit = "banane";
  }
  getFruit() {
    console.log(this.fruit);
  }
}

// Dans le fichier Barquette.js
import { Panier } from './Panier.js';

class Barquette extends Panier {
  constructor() {
    super();
    this.maxq = 5;
  }
}

let barquette = new Barquette();
console.log(barquette); // {fruit: "banane", maxq: 5}
```




Exercice 1:

module Personnage

Créer un module Personnage qui contiendra les classes:

 **Personnage**

 **Aventurier**

 **Guerrier**

 **Paladin**

 **Mage**



La manipulation du DOM en JavaScript

Le DOM (Document Object Model) est une interface de programmation pour les documents HTML, XML et SVG. Il fournit une représentation structurée du document, permettant ainsi à des programmes de modifier la structure, le style et le contenu du document.








La manipulation du DOM en JavaScript

L'objet `document` représente le document HTML. Il permet d'accéder à tous les éléments du document et de les manipuler.



Pour manipuler des éléments du DOM, on utilise les méthodes suivantes:

-  `document.getElementById(id)` : Renvoie un élément du DOM en fonction de son identifiant
-  `document.getElementsByClassName(nom)` : Renvoie une liste d'éléments du DOM en fonction de leur classe
-  `document.getElementsByTagName(nom)` : Renvoie une liste d'éléments du DOM en fonction de leur nom de balise
-  `document.querySelector(selecteur)` : Renvoie le premier élément du DOM correspondant au sélecteur CSS
-  `document.querySelectorAll(selecteur)` : Renvoie une liste d'éléments du DOM correspondant au sélecteur CSS



Manipulation des éléments du DOM en JavaScript

Exemple:

```
let titre = document.getElementById("titre");  
let paragraphes = document.getElementsByClassName("paragraphe");  
let paragraphes = document.getElementsByTagName("p");  
let titre = document.querySelector("#titre");  
let paragraphes = document.querySelectorAll(".paragraphe");
```




Les événements en JavaScript


Un événement est une action que l'utilisateur peut effectuer sur un élément du DOM. Par exemple, l'utilisateur peut cliquer sur un bouton, remplir un champ de formulaire, etc.



Les événements en JavaScript

Pour détecter un événement, on utilise les méthodes suivantes:

 `element.addEventListener(nom, fonction)` : **Ajoute un événement à un élément du DOM**

 `element.removeEventListener(nom, fonction)` : **Supprime un événement d'un élément du DOM**



Les événements en JavaScript

Exemple:

```
let bouton = document.getElementById("bouton");  
bouton.addEventListener("click", function() {  
    console.log("Le bouton a été cliqué");  
});
```




Les événements en JavaScript

Exemple:

```
let bouton = document.getElementById("bouton");  
function afficherMessage() {  
    console.log("Le bouton a été cliqué");  
}  
bouton.addEventListener("click", afficherMessage);  
bouton.removeEventListener("click", afficherMessage);
```



Conclusion

Nous avons vu:

- 📌 **Les types et portées de données**
- 📌 **Les fonctions**
- 📌 **La création d'objets et l'héritage**
- 📌 **La manipulation du DOM et les événements**



Documentation

 **MDN**

 **W3Schools**