PROGRAMMATION ORIENTÉE OBJET ET WEB

Pr. OMARI Kamal

FACULTE POLYDISCIPLINAIRE D'OUARZAZATE

23 mars 2025

Chapitre 4 : Exploration du Langage JS

Objectifs du cours

Dans ce chapitre, nous allons ...

Notion de compilateur / interpréteur

La compilation consiste à transformer le code écrit dans un langage de programmation de haut niveau (code lisible par l'homme) en code machine compréhensible par un processeur informatique (bits binaires 1 et 0). Le compilateur s'assure également que le programme est correct du point de vue TYPE : on n'est pas autorisé à affecter une chaîne de caractères à une variable entière. Le compilateur veille également à ce que votre programme soit syntaxiquement correct. Par exemple, "x * y" est valide, mais "x @ y" ne l'est pas.

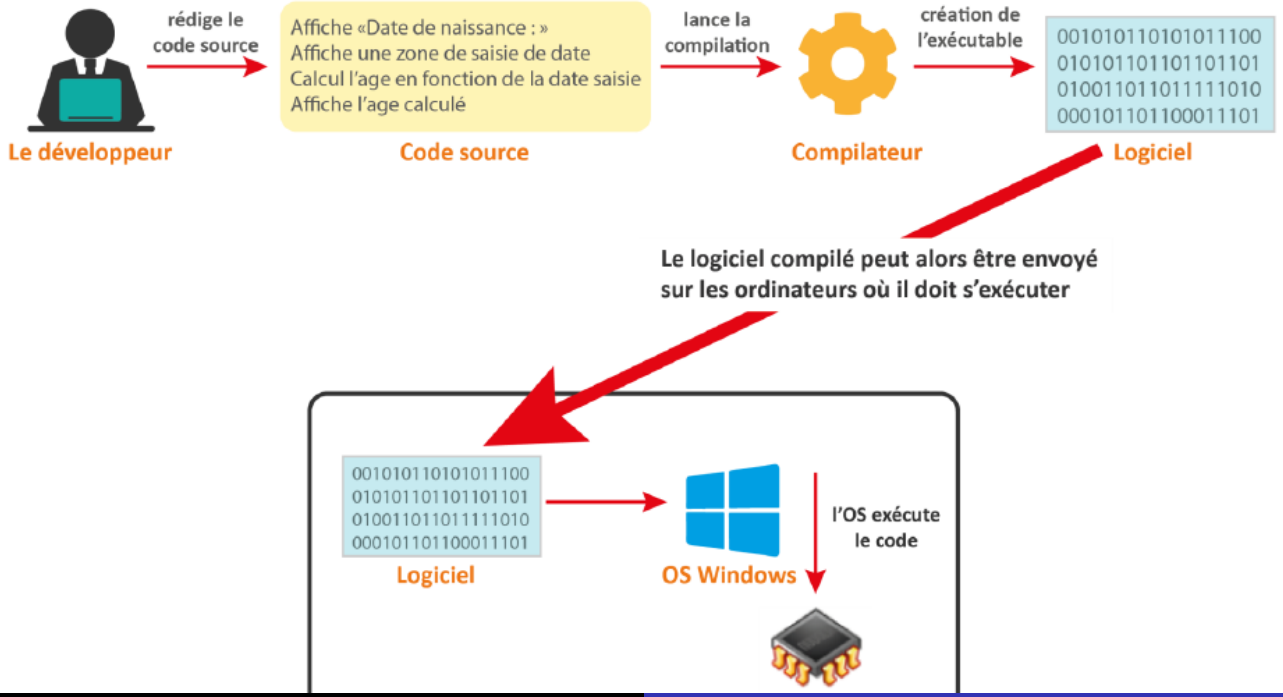
Notion de compilateur / interpréteur

Un interpréteur est un programme informatique qui convertit chaque déclaration de programme de haut niveau en code machine. Cela inclut le code source, le code précompilé et les scripts.

Notion de compilateur / interpréteur

Différence : Un compilateur convertit le code en code machine (crée un exe) avant l'exécution du programme. L'interpréteur convertit le code en code machine, ligne par ligne, au moment d'exécution du programme.
Exemples de langages compilés : C, C++, C#, CLEO, COBOL, etc.
Exemples de langages interprétés : JavaScript, Perl, Python, BASIC, etc.

Langage compilé



Langage interprété

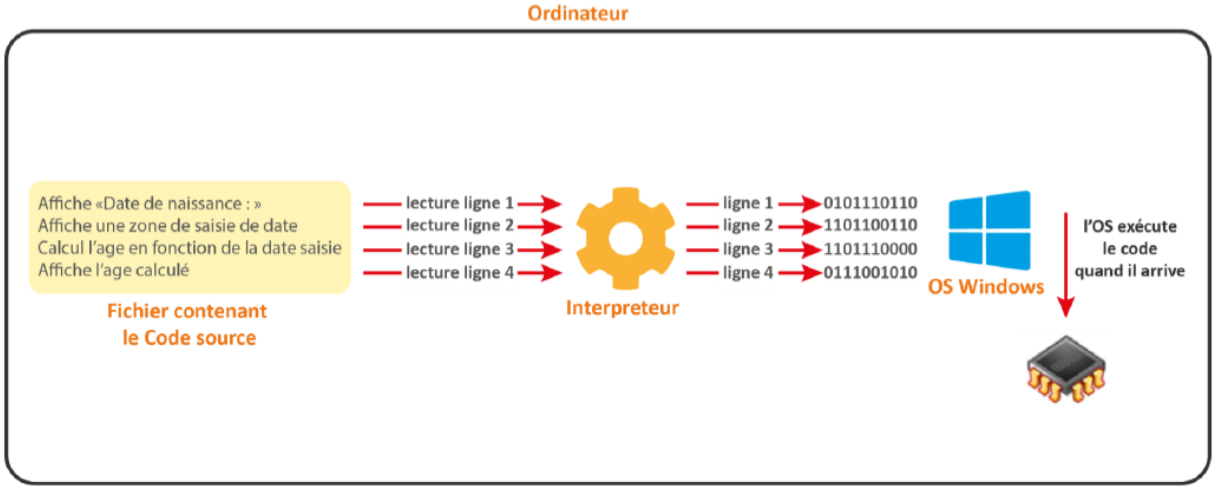


Figure 2 – Fonctionnement d'un langage interprété

Langage de script

- Un langage de script (également appelé script) est une série de commandes qui peuvent être exécutées sans compilation.
- Tous les langages de script sont des langages de programmation, mais tous les langages de programmation ne sont pas des langages de script.
- Les langages de script utilisent un programme appelé interpréteur pour traduire les commandes.

Types d'un langage de script : coté client et coté serveur

Les langages de script côté serveur s'exécutent sur un serveur Web Lorsqu'un client envoie une requête, le serveur répond en envoyant du contenu via le protocole HTTP Les scripts côté serveur ne sont pas visibles par le public Leur rôle est d'assurer la rapidité du traitement, l'accès aux données et la résolution des erreurs.
Exemples de langages de script côté serveur :
PHP, ASP NET, Node Js Java, Ruby, Perl.

Types d'un langage de script : coté client et coté serveur

Les langages de script côté client s'exécutent du côté du client, sur son navigateur Web L'avantage des scripts côté client est qu'ils peuvent réduire la demande sur le serveur, ce qui permet aux pages Web de se charger plus rapidement Ces scripts sont axés sur l'interface utilisateur et ses fonctionnalités.
Exemples de langages de script côté client :
HTML, CSS, JavaScript.

Le rôle de l'interpréteur

L'interpréteur exécute les langages de script en traduisant les instructions du programme source en code machine. En cas d'erreur dans une instruction, il interrompt la traduction, affiche un message d'erreur et ne passe à l'instruction suivante qu'une fois l'erreur corrigée. Contrairement aux compilateurs, l'interpréteur exécute directement les instructions sans les convertir d'abord en code objet ou en code machine.

Le rôle de l'interpréteur

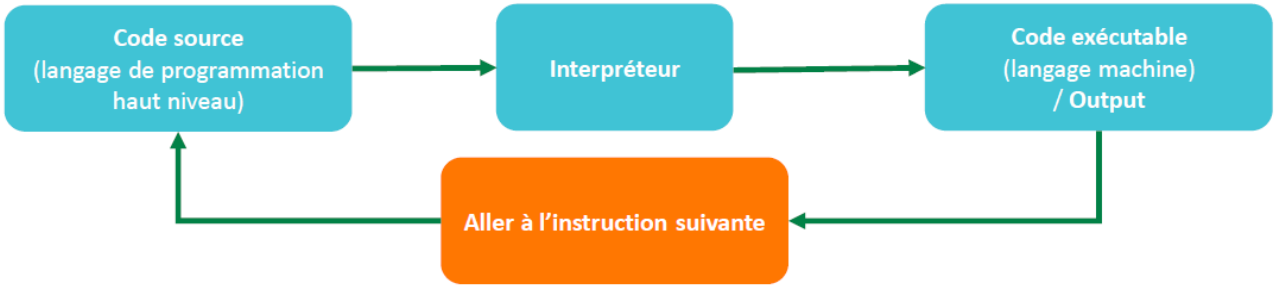


Figure 3 – Fonctionnement d'un langage de script

Définition de l'architecture Client / Serveur

L'architecture client serveur correspond à l'architecture d'un réseau informatique dans lequel de nombreux clients (processeurs distants) demandent et reçoivent des services d'un serveur centralisé (Serveur).
Les clients sont souvent situés sur des postes de travail ou des ordinateurs personnels, tandis que les serveurs sont situés ailleurs sur le réseau, généralement sur des machines plus puissantes

Définition de l'architecture Client / Serveur

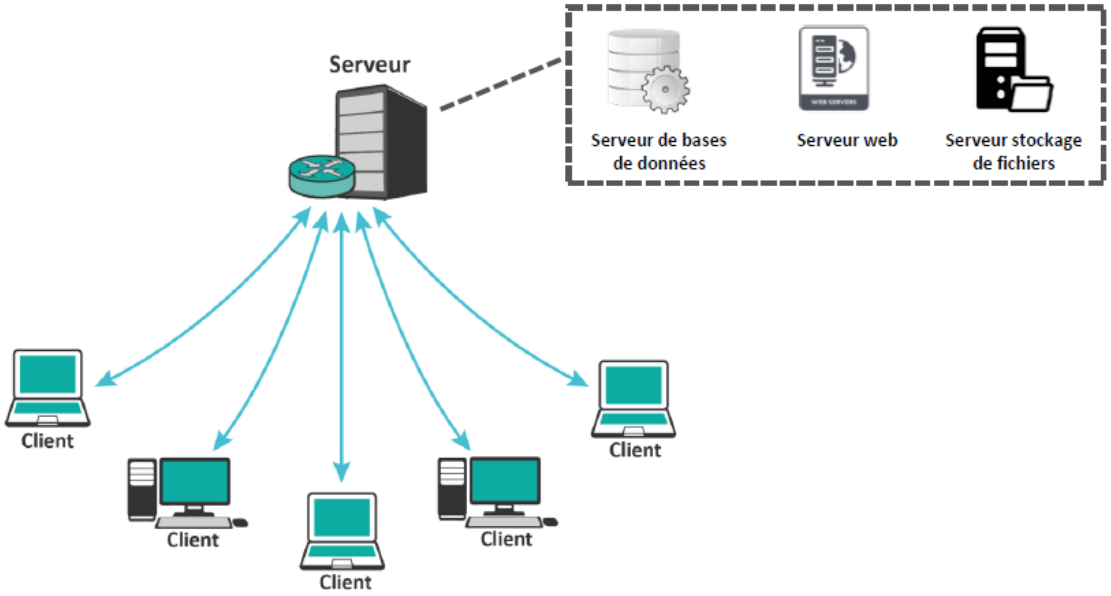


Figure 4 – Architecture Client serveur

Interaction Client / Serveur

Les ordinateurs clients fournissent une interface (comme les navigateur) permettant à un utilisateur de demander des services auprès de serveur et d'afficher les résultats. Cette interaction passe par les étapes suivantes :

- L'utilisateur saisit l'URL (Uniform Resource Locator) du site web ou du fichier. Le navigateur demande alors au serveur le DNS (DOMAIN NAME SYSTEM).
- Le serveur DNS recherche l'adresse du serveur WEB.
- Le serveur DNS répond avec l'adresse IP du serveur Web.
- Le navigateur envoie une requête HTTP/HTTPS à l'adresse IP du serveur WEB (fournie par le serveur DNS).
- Le serveur envoie les fichiers nécessaires du site web (pages html, documents, images, ...).
- Le navigateur rend alors les fichiers et le site web s'affiche. Ce rendu est effectué à l'aide de l'interpréteur DOM (Document Object Model),de l'interpréteur CSS et du moteur JS, collectivement connus

Fonctionnement

Le fonctionnement d'un système client/serveur peut être illustré par le schéma suivant :

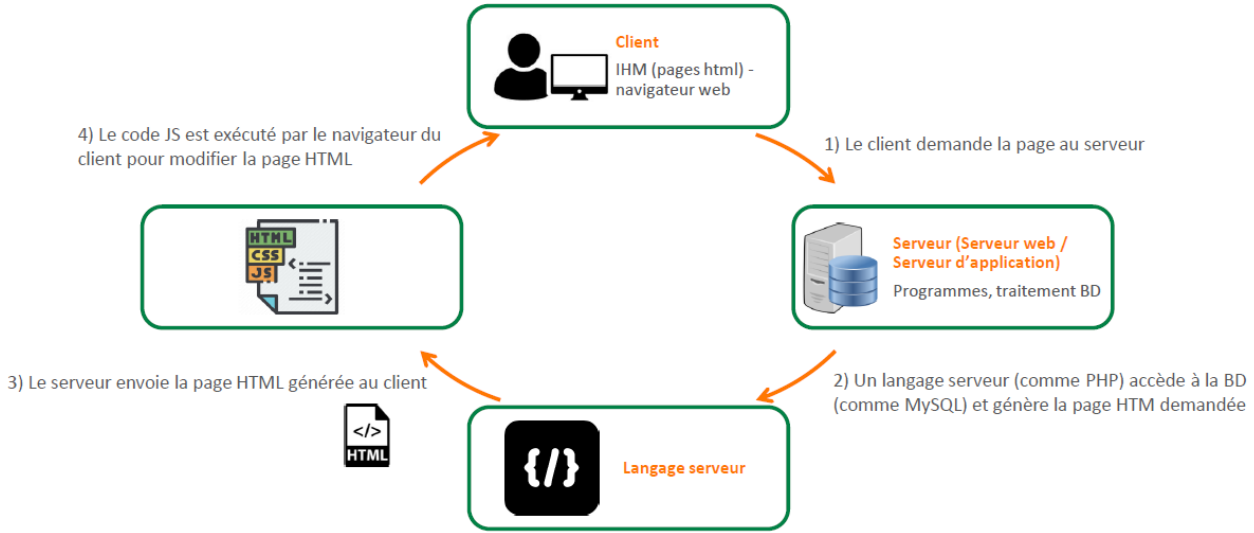


Figure 5 – Fonctionnement d'un système client serveur

Introduction à JavaScript

- Javascript est un langage de scripts qui peut être incorporé aux balises Html. JS est exécuté par le navigateur ;
- Son rôle est d'améliorer la présentation et l'interactivité des pagesWeb ;
- JavaScript offre des « gestionnaires d'événement » qui reconnaissent et réagissent aux demandes du client (comme les mouvements de la souris, clics sur les touches du clavier, etc.)

Remarques

- JS est un langage de Script dépendant de HTML.
- Code interprété par le browser au moment de l'exécution à la volée (Just In Time).
- JS Permet d'accéder aux objets du navigateur.
- JavaScript est « case sensitive ».

Intégration de JavaScript dans HTML

JavaScript peut être intégré n'importe où dans le document HTML Il est aussi possible de l'utiliser plusieurs fois dans le même document HTML

Méthode 1 Code JavaScript intégré au document html

```
<script language ="JavaScript">
/* ou // code js
</script>
```

Méthode 2 Code JavaScript externe

```
<script language ="javascript" src="monScript.js">
</script>
```

Identifiants JavaScript

- Un identifiant en JS correspond au nom d'une variable Ce nom doit être unique.
- Ces identifiants peuvent être des noms courts (comme x et y) ou bien des noms plus descriptifs (comme note, total, NomComplet etc.

Les règles à respecter lors du choix des noms de variables sont :

- `Un identifiant peut contenir des lettres, des chiffres, des traits de soulignement (_) et des signes dollar ($).`
- `Un identifiant peut commencer par une lettre, un signe $ ou bien un signe _`
- `JS est sensible à la casse (y et Y sont considérés comme des variables différentes).`
- `Un identifiant ne peut pas être un mot réservé du langage.`

Types de données JavaScript

JavaScript est un langage faiblement typé : le type d'une variable est défini au moment de l'exécution.
On peut déclarer une variable JS en utilisant les mots-clés suivants :

- var : utilisé pour déclarer une variable globale (function scope) ;
- let : utilisé pour déclarer une variable dont la portée est limitée à un bloc (block scope) ;
- const : permet de déclarer une variable qui doit avoir une valeur initiale et ne peut pas être réassignée.

Types de données JavaScript (exemples)

Déclaration des variable booléennes :

- var test=new Boolean(true) ;
- test=new Boolean(1) ;
- let test = true.

Déclaration des chaînes de caractères :

- var chaine = "Bonjour" ;

Déclaration des nombres :

- var entier = 60 ; //un entier ;
- let pi = 3.14 ; //un nombre réel.

Déclaration de plusieurs variables :

- var variable3 = 2, variable4 = "mon texte d'initialisation" ;

Déclaration sans initialisation :

- Une variable déclarée sans valeur aura la valeur undefined.

Types de données JavaScript (exemples)

Const permet de créer des variables JavaScript qui ne peuvent être ni redéclarées ni réaffectées (Ces variables doivent être initialisées à la déclaration.

Exemple 1

```
const PI 3.141592653589793;
PI = 3.14 ;;// Erreur
PI = 10 ;;// Erreur
```

Exemple 2

```
const PI;
PI = 3.14159265359 ; // Incorrect
```

Portée des variables

La portée d'une variable détermine son accessibilité (visibilité).
En JS, trois types de portées sont distinguées :

Portée du bloc : (Block scope)

```
En utilisant le mot clé let, les variables déclarées à
l'intérieur d'un bloc   ne sont pas accessibles depuis
l'extérieur du bloc :
{let x = 2; }// x n'est pas accessible ici
En utilisant le mot clé var, les variables déclarées
à l'intérieur d'un bloc   sont accessibles depuis
l'extérieur du bloc.
{var x = 2; }// x est accessible ici
```

Portée des variables

Les variables déclarées dans une fonction JavaScript deviennent LOCALES à la fonction Ils ne sont accessibles qu'à partir de la fonction.

Portée locale : :(Function scope)

```
function Test()
{
var x ="test1" ;
let y = "test2";
const z = "test3";
}
//x, y et z ne sont pas accessibles en dehors de la
fonction
```

Portée des variables

Portée globale : Une variable déclarée en dehors d'une fonction, devient GLOBAL Les variables globales sont accessibles de n'importe où dans un programme JavaScript.

Opérateurs arithmétiques JavaScript

Les opérateurs arithmétiques sont utilisés pour effectuer des opérations arithmétiques sur des nombres.

Opérateur	**Signification**
+	Addition
-	Soustraction
*	Multiplication
**	Puissance
/	Division
%	Modulo (Reste de la division)
++	Incrémentation
–	Décrémentation

Table 1 – Tableau des opérateurs et leur signification

Opérateurs d'affectation JavaScript

Les opérateurs d'affectation permettent d'assigner des valeurs aux variables JavaScript Le tableau suivant regroupe ces opérateurs.

Exemple d'utilisation	Signification	Exemple complet	Résultat
`x = y`	x prend la valeur de y	`let x = 5;`	x vaut 5
`x += y`	x = x + y	`let x = 10; x += 5;`	x vaut 15
`x -= y`	x = x - y	`let x = 10; x -= 5;`	x vaut 5
`x *= y`	x = x * y	`let x = 10; x *= 5;`	x vaut 50
`x /= y`	x = x / y	`let x = 10; x /= 5;`	x vaut 2
`x %= y`	x = x % y	`let x = 10; x %= 5;`	x vaut 0
`x **= y`	x = x à la puissance y	`let x = 3; x **= 2;`	x vaut 9

Concaténation des chaînes de caractères en JavaScript

L'opérateur +, appliqué aux chaînes de caractères, permet de concaténer des chaînes.

Exemple

```
let texte1 ="FP";
texte1 +=" ";
let texte2 ="Ouarzazate";
let texte3 = texte1 + texte2;
//Output : texte3 = "FP Ouarzazate"
```

Concaténation des chaînes de caractères en JavaScript

L'application de l'opérateur + pour concaténer les chaines de caractères et les nombres :

Exemple

```
let x = 1 + 1;
let y = "5" + 1;
//x=2
//y="51"
```

Opérateurs de comparaison en JavaScript

Les opérateurs de comparaison permettent de comparer des opérandes (qui peuvent être des valeurs numériques ou des chaînes de caractères) et renvoie une valeur logique : true (vrai) si la comparaison est vraie, false (faux) sinon.

Frame Title

Opérateur	Signification	Exemple
==	Égal à (comparaison des valeurs)	`let x = 5; let y = "5"; let z = (x == y); // z = true`
===	Égal à (comparaison de la valeur et du type)	`let x = 5; let y = "5"; let z = (x === y); // z = false`
!=	Différent de (n'est pas égal à)	`let x = 5; let y = 5; let z = (x != y); // z = false`
!==	Type ou valeur différente	`let x = 5; let y = "5"; let z = (x !== y); // z = true`
>	Supérieur à	`let x = 5; let y = 5; let z = (x > y); // z = false`
<	Inférieur à	`let x = 5; let y = 5; let z = (x < y); // z = false`
>=	Supérieur ou égal à	`let x = 5; let y = 5; let z = (x >= y); // z = true`
<=	Inférieur ou égal à	`let x = 5; let y = 5; let z = (x <= y); // z = true`

Table 3 – Tableau des opérateurs de comparaison en JavaScript

Opérateurs logiques en JavaScript

Les opérateurs logiques, aussi appelés opérateurs booléens, sont utilisés pour combiner des valeurs booléennes (des conditions qui peuvent avoir les valeurs true ou false) et produire une nouvelle valeur booléenne.

Frame Title

Opérateur	Signification
&&	ET logique
||	OU logique
!	NON logique

Table 4 – Tableau des opérateurs logiques

Possibilités d'affichage JavaScript

JavaScript permet d'afficher les données de différentes manières :

- document.write() : Écriture dans le document de sortie HTML.
- window.alert() : Écriture dans une boîte d'alerte.
- console.log() : Écriture dans la console du navigateur. Cette méthode est pratique pour le débogage du code.

document.write()

La méthode document.write() en JavaScript permet d'écrire directement dans le document HTML. Lorsqu'elle est utilisée, elle insère du texte ou du contenu HTML dans le flux de sortie de la page.

Exemple

```
document.write("Ceci est un message affiché dans le
document HTML.");
document.write("<h1>Bonjour, bienvenue sur notre site
!</h1>");
```

window.alert()

La méthode window.alert() en JavaScript affiche une boîte de dialogue contenant un message et un bouton "OK". Elle est souvent utilisée pour avertir l'utilisateur ou lui donner des informations simples, mais comme elle bloque l'interaction avec la page jusqu'à ce que l'utilisateur clique sur "OK", elle est rarement utilisée dans les applications modernes où des méthodes plus élégantes sont privilégiées.

window.alert()

Exemple

```
window.alert("Ceci est une alerte !");
```

console.log()

La méthode console.log() en JavaScript permet d'afficher des informations dans la console du navigateur, utile pour le débogage et le suivi de l'exécution de votre code. Contrairement à alert(), elle n'interrompt pas le flux de votre application et est invisible pour les utilisateurs finaux.

console.log()

Exemple

```
console.log("Message à afficher dans la console");
```

Impression JavaScript

La méthode window.print() permet d'imprimer le contenu de la fenêtre en cours en appelant le dispositif propre du navigateur. L'exemple suivant permet d'appeler la méthode « window.print() » suite au clic sur un bouton.

Exemple

```
<!DOCTYPE html>
<html>
<body>
<button onclick="window.print()">Imprimer cette
page</button>
</body>
</html>
```

Les entrées Javascript

En JavaScript, on peut récupérer les données de deux manières différentes :

- Prompt()
- document.getElementById(id)

Les entrées Javascript

Prompt : affiche une boîte de dialogue avec une zone de saisie

- La méthode prompt (boite d'invite) propose un champ comportant une entrée à compléter par l'utilisateur avec une valeur par défaut.
- En cliquant sur OK, la méthode renvoie la valeur tapée par l'utilisateur ou la réponse proposée par défaut. Si l'utilisateur clique sur Annuler (Cancel), la valeur null est alors renvoyée.

Exemple

```
var prenom = prompt("Quel est votre prénom?");
document.write(prenom);
```

Les entrées Javascript

Les formulaires : les contrôles des formulaires comme la zone <input>

- Javascript peut récupérer les données de n'importe quel élément html par la méthode : document.getElementById(id) qui prend en paramètre l'identifiant de l'objet.

Exemple

```
<!DOCTYPE html>
<html>
<body>
<input type="text" id="prenom">
<button onclick="alert(document.getElementById
('prenom').value)">Afficher</button>
</body>
</html>
```

Types primitifs et objets de base

En JavaScript, on distingue deux familles de types de variables

Types primitifs

- String : chaines de caractères
- Number : nombres entiers et réels
- Boolean : les booléens (rue, false)
- Undefined : valeur prise par les variables déclarées non initialisées

Les types structurels

- Date : pour représenter les dates
- Array : pour représenter les tableaux
- Object : tel que les enregistrements
- Function

La déclaration if

L'instruction if est utilisée pour spécifier un bloc de code JavaScript à exécuter si une condition est vraie.

Syntaxe

```
if(condition)
{
//bloc d'instructions à executer si la condition est vraie
}
```

Exemple
On affiche "Réussi" si la note est supérieure ou égale à 10.

Syntaxe

```
if(note >= 10)
{
document.write ("Réussi")
```

La déclaration else

L'instruction else est utilisée pour spécifier un bloc de code à exécuter si la condition est fausse.

Syntaxe

```
if (condition)
{
// bloc d'instructions à executer si la condition est
vraie.
}
else
{
//bloc d'instructions à executer si la condition est
fausse.
}
```

L'instruction else if

L'instruction else if est utilisée pour spécifier une nouvelle condition si la première condition est fausse.

Syntaxe

```
if (condition1)
//bloc d'instructions à executer si la condition1 est
vraie.
}
else if (condition2)
// bloc d'instructions à executer si la condition2 est
vraie et la condition1 est fausse. }
else
//bloc d'instructions à executer si la condition2 est
fausse.
}
```

L'instruction switch

L'instruction switch est utilisée pour sélectionner un choix parmi plusieurs (pour remplacer les blocs if else multiples).

Syntaxe

```
switch (expression)
case x:
// code block
break;
case y:
// code block
break; default:
// code block
}
```

Structures itératives

JavaScript, comme la plupart des langages de programmation, prend en charge les types de boucles suivantes :

- for : parcourt un bloc de code plusieurs fois.
- for/in : parcourt les propriétés d'un objet.
- for/of : parcourt les valeurs d'un objet itérable.
- while : parcourt un bloc de code tant qu'une condition spécifiée.
- do/ while : parcourt un bloc de code tant qu'une condition spécifiée est vraie. Exécute les instructions au moins une seule fois.

La boucle For

Syntaxe

```
for (let i = 0; i < 5; i++) {
document.write("Le nombre est " + i + "<br>");
}
```

La boucle For in

Syntaxe

```
const person = {fname: "FP", lname: "Ouarzazate",
Annee: 2024 };
for (let key in person) {
console.log(key + ": " + person[key]);
}
```

La boucle For of

La boucle for...of en JavaScript est utilisée pour parcourir les valeurs d'un objet itérable, tel qu'un tableau, une chaîne de caractères, une liste de nœuds, ou tout autre objet qui implémente l'interface Iterable.

Exemple

```
const fruits = ["Pomme", "Banane", "Cerise"];
for (const fruit of fruits) {
console.log(fruit);
}
```

Différence entre for...of et for...in

- for...of parcourt les valeurs d'un objet itérable.
- for...in parcourt les propriétés (clés) d'un objet.

Exemple

```
const numbers = [10, 20, 30];
console.log("Avec for...in :");
for (const index in numbers) {
console.log(index); // Affiche les indices : 0, 1, 2
}
console.log("Avec for...of :");
for (const value of numbers) {
console.log(value); // Affiche les valeurs : 10, 20,
30
```

La boucle while

La boucle while en JavaScript permet de parcourir un bloc de code tant qu'une condition spécifiée est vraie. Tant que la condition reste vraie, le code à l'intérieur de la boucle sera exécuté.

Syntaxe

```
while (condition) {
// bloc de code à exécuter tant que la condition est
vraie
}
```

Exemple

Exemple

```
let i = 0;
while (i < 5) {
console.log(i); // Affiche les valeurs de i de 0 à 4
i++; // Incrémente i de 1 à chaque itération
}
```

La boucle do...while

La boucle do...while en JavaScript fonctionne de manière similaire à la boucle while, mais avec une différence importante : elle exécute le bloc de code au moins une fois, même si la condition est fausse dès le début.

Syntaxe

```
do {
// bloc de code à exécuter
}while (condition);
```

Exemple

Exemple

```
let i = 0;
do {
console.log(i); // Affiche la valeur de i
i++; // Incrémente i de 1 à chaque itération
}while (i < 5);
```

La déclaration Break

L'instruction break en JavaScript est utilisée pour sortir immédiatement d'une boucle (comme une boucle for, while, ou do...while) ou d'un switch.

Exemple

```
for (let i = 0; i < 5; i++) {
if (i === 3) {
break; // La boucle s'arrête lorsque i vaut 3
}
console.log("Le compteur est :", i);
}
```

La déclaration continue

L'instruction continue en JavaScript est utilisée pour sauter l'itération courante d'une boucle si une condition spécifiée est vraie, et passe directement à l'itération suivante.

Exemple

```
for (let i = 0; i < 5; i++) {
if (i === 3) {
continue; // Ignore l'itération lorsque i vaut 3
}
console.log("Le compteur est :", i);
}
```

Définition

- Une fonction est définie comme un bloc de code organisé et réutilisable, créé pour effectuer une action unique.
- Le rôle des fonctions est d'offrir une meilleure modularité au code et un haut degré de réutilisation.
- Une fonction JavaScript est un ensemble d'instructions qui peut prendre des entrées de l'utilisateur, effectuer un traitement et renvoyer le résultat à l'utilisateur.
- Une fonction JavaScript est exécutée au moment de son appel.

Définition

En JavaScript, il existe 2 types de fonctions :

- Les fonctions propres à JavaScript : appelées "méthodes". Elles sont associées à un objet bien particulier comme la méthode alert () avec l'objet window .
- Les fonctions écrites par le développeur : déclarations placées dans l'en tête de la page.

Syntaxe des fonctions en JavaScript

- En JavaScript, une fonction est définie avec le mot clé function suivi du nom de la fonction et des parenthèses.
- Le nom de la fonction doit respecter les mêmes règles que les noms des variables.
- Les parenthèses peuvent inclure des noms de paramètres séparés par des virgules Les arguments de la fonction correspondent aux valeurs reçues par la fonction lorsqu'elle est invoquée.
- Le code à exécuter, par la fonction, est placé entre accolades.

Syntaxe

```
function nomFonction (paramètre1 paramètre2 paramètre3, ...)
{
// Code de la function
return ... ;
}
```

Retour de la fonction

- L'instruction return est utilisée pour renvoyer une valeur (souvent calculée) au programme appelant.
- Lorsque l'instruction return est atteinte, la fonction arrête son exécution.

Appel de fonction

Le code de la fonction est exécuté quand la fonction est appelée selon les cas suivants :

- Lorsqu'un événement se produit (clic sur un bouton par exemple) ;
- Lorsqu'il est invoqué (appelé) à partir d'un autre code JavaScript ;
- Automatiquement (auto invoqué).

La fonction est appelée en utilisant son nom et, si elle prend des paramètres, en indiquant la liste de ses arguments :

- nomFonction (p1, p2, p3, ...)

Exemple

Exemple

```
let x = myFunction(4, 3); // La fonction est appelée,
la valeur retournée est affectée à x
// Définition de la fonction
function myFunction(a, b) {
return a * b; // La fonction retourne le produit de a
et b
}console.log(x); // Affiche le résultat : 12
```

Les expressions lambdas

- Les fonctions fléchées (arrow functions) sont des fonctions qui ont une syntaxe compacte. Par conséquent, elles sont plus rapide à écrire que les fonctions traditio nne lles ;
- Les fonctions fléchées sont limitées et ne peuvent pas être utilisées dans toutes les situations ;
- Principe : à la place du mot clé function , on utilise le signe (=>) plus une parenthèse carrée fermante (>) après la parenthèse fermante de la fonction.

Exemple

Exemple

```
// Déclaration d'une fonction avec une variable
const maFonction = () =>{
return "ma variable";
}
// Appel de la fonction et affichage du résultat
console.log(maFonction()); // Affiche : Hello, World!
```

Exemple 2

Exemple 2

```
// Déclaration d'une fonction avec une variable
const maFonction = (a,b) =>{
return a*b;
}
// Appel de la fonction et affichage du résultat
console.log(maFonction(2, 3)); // Affiche : 6
```

Appels asynchrones (CallBack, Promise)

- Les fonctions JavaScript sont exécutées dans l'ordre où elles sont appelées Pas dans l'ordre où elles sont définies.
- Une fonction peut appeler une fonction dans son code, directement ou en utilisant les fonctions de rappel.
- Une fonction de rappel (callback en anglais) est une fonction passée en paramètre d'une autre fonction en tant qu'argument.
- La fonction de rappel est invoquée à l'intérieur de la fonction externe pour exécuter des instructions précises.

Exemple

Exemple

```
function affichage(some) {
document.write(some);
}
function myCalculator(num1, num2, myCallback) {
let somme = num1 + num2;
myCallback(somme);
}
myCalculator(5, 5, affichage);
```

Les promesses (Promise)

- Une promesse (Promise) est un objet JavaScript qui contient à la fois le code producteur et les appels au code consommateur : renvoie une valeur q u'o n va utiliser dans le futur.
- La promesse est adaptée à la gestion des opérations asynchrones.

Exemple

```
Instruction1
Instruction2
Instruction3
//Qui récupère une valeur externe et la mettre dans un élément HTML
(cela prend 5 secondes)
Instruction4
```

Problème : Le code actuel provoque une attente de 5 secondes avant d'exécuter l'instruction 4
Solution : en utilisant un objet Promise, on peut provoquer l'exécution de l'instruction 3 et en même temps continuer l'exécution du programme Quand l'instruction 3 récupère la valeur depuis la ressource externe, elle sera placée dans

Les promesses (Promise)

En JavaScript, une promesse a trois états :

- En attente (Pending) : résultat indéfini ;
- Accompli (Fulfilled) opération réussie, le résultat est une valeur ;
- Rejeté (Rejected) le résultat est une erreur.

Une promesse commence toujours dans l'état « en attente » et se termine par un des états « accompli » ou « rejeté » comme illustré sur la figure suivante

Exemple

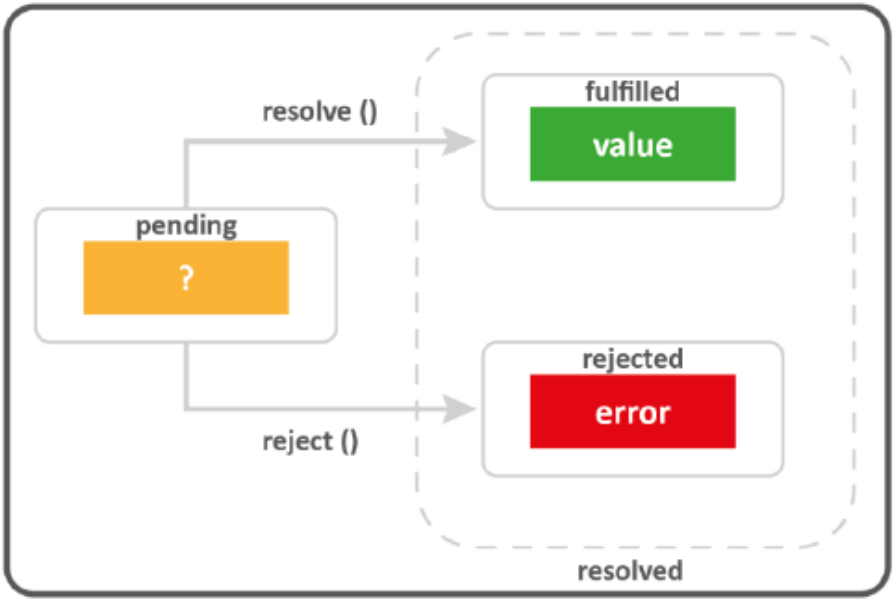


Figure 6 – Les états d'une promesse

Créer une promesse : le constructeur Promise

Pour créer une promesse en JavaScript, on utilise le constructeur Promise

Exemple

```
let completed = true;
let getData = new Promise(function (resolve, reject) {
if (completed) {
resolve("Donnée récupérée"); //Appelle resolve si completed est vrai
}else {reject("Je n'ai pas pu récupérer la donnée");//Appelle reject si
completed est faux
}
});
// Utilisation de la promesse
getData
.then((message) => {console.log(message); //Affiche : "Donnée récupérée"
si la promesse est résolue
})
.catch((erreur) => {console.error(erreur); //Affiche : "Je n'ai pas pu
récupérer la donnée" si la promesse est rejetée
});
```

Consommer une promesse :then , catch, finally

- .then() pour gérer le succès.
- .catch() pour gérer l'échec.
- .finally() pour exécuter du code, qu'il y ait une erreur ou non.

Le bloc try ... catch

Lorsqu'une erreur se produit, JavaScript arrête et crée un objet d'erreur error avec deux propriétés nom et un message (varie selon le navigateur). JavaScript lance une exception, ont dit qu'une erreur est générée. Pour gérer les exceptions en JavaScript on utilise les mots clés suivants :

- try tester un bloc de code pour les erreurs
- catch gérer l'erreur
- throw créer des erreurs personnalisées
- Finally permet l'exécution de code, après try et catch, quel que soit le résultat

Exemple

Exemple

```
try {
//Bloc du code à exécuter
}
catch (erreur){
// Bloc du code qui gère l'exception
}
```

L'instruction throw

En JavaScript, l'instruction throw est utilisée pour lancer une exception. Cela signifie que vous pouvez créer une erreur personnalisée et l'envoyer dans votre programme, ce qui interrompt l'exécution normale du code et passe le contrôle à une structure de gestion des erreurs comme try...catch.

Exemple

Exemple

```
let x = Number(prompt("Donnez un numéro entre 5 et 10 :"));
try {
// Vérifier si l'entrée est vide
if (x == "") throw "Vide";
// Vérifier si l'entrée n'est pas un nombre
if (isNaN(x)) throw "Ce n'est pas un numéro";
// Convertir en nombre (inutile ici car prompt retourne déjà un nombre
avec Number())
x = Number(x);
// Vérifier si le nombre est trop petit
if (x < 5) throw "Trop petit";
// Vérifier si le nombre est trop grand
if (x > 10) throw "Trop grand";
console.log("Numéro valide :", x); // Affiche le numéro valide
}catch (err) {
console.error("Erreur :", err); // Affiche le message d'erreur
```

L'objet error

L'objet error de JavaScript fournit des informations sur l'erreur quand elle se produit Cet objet fournit deux propriétés name et message.

Table 5 – Valeurs de nom d'erreur

Nom de l'erreur	**Description**
EvalError	Erreur dans la fonction `eval()`
RangeError	Nombre hors limite
ReferenceError	Référence inconnue
SyntaxError	Erreur de syntaxe
TypeError	Une erreur de type s'est produite
URIError	URI incorrecte

Exemple

Exemple

```
let x = 5;
try {
// Tentative d'utiliser une variable non définie
x = y + 1; // 'y' n'est pas référencée en mémoire
}catch (err) {
console.log(err.name); // Affiche : "ReferenceError"
console.log(err.message); // Affiche le message
d'erreur spécifique
}
```

Création d'objet avec syntaxe littérale

La syntaxe littérale consiste à créer un objet en définissant ses propriétés à l'intérieur d'une paire d'accolades : { ... }.

Syntaxe

```
const monObjet = {
propriete1: "valeur1",
propriete2: "valeur2",
// Ajoute une virgule entre les propriétés et méthodes
methode1(/* paramètres */) {
// Code de la méthode 1
},
methode2(/* paramètres */) {
// Code de la méthode 2
}
};
monObjet.methode1(); // Appel de la méthode 1
```

Remarque

- Une méthode est une propriété dont la valeur est une fonction Son rôle est de définir un comportement (Action) pour l'objet.
- On peut utiliser var au lieu de const.

Exemple

Exemple

```
// Définition de l'objet 'telephone'
const telephone = {
marque: "SmartF22",
prix: 400,
stock: 200,
ref: "Smartphone 2024",
// Méthode pour vérifier le stock
VerifierStock() {
if (this.stock > 0) {
return true;
}else {
return false;
}
}
};
// Test du code
console.log(telephone.marque); // Affiche : SmartF22
console.log(telephone.VerifierStock()); // Affiche : true
```

Création d'objet avec constructeur

La création des objets en utilisant un constructeur permet d'instancier cet objet dans des variables différentes.

Syntaxe

```
// Définition de la fonction constructeur
function MonObjet(param1, param2) {
this.propriete1 = param1;
this.propriete2 = param2;
// Définition d'une méthode
this.methode1 = function() {.....}; }// Création d'une instance
de l'objet
const monObjet = new MonObjet("Valeur 1", "Valeur 2");
// Appel de la méthode
monObjet.methode1();
```

Exemple

Syntaxe

```
// Définition de la fonction constructeur
function Telephone(n, p, s, r) {
this.nom = nom;
this.prix = prix;
this.stock = stock;
this.ref = ref;
// Méthode pour vérifier le stock
this.VerifierStock = function() {
if (this.stock > 0) {
return true;
}else {
return false;
}
};
}
// Création des instances de Telephone
var t1 = new Telephone("SmartF22", 400, 200, "pro1");
var t2 = new Telephone("Mi Max", 200, 0, "t2");
// Test des propriétés et méthodes
console.log(t1.nom); // Affiche : SmartF22
console.log(t2.VerifierStock()); // Affiche : false
```

Ajouter/modifier des propriétés ou des méthodes

On peut ajouter des méthodes et des propriétés aux objets crées
telephone.dateSortie ='2022' ;

Supprimer une propriété ou une fonction

On supprime une propriété en utilisant le mot clé delete : delete telephone.dateSortie.

Les tableaux : déclaration

Un tableau en JavaScript est un objet qui hérite de l'objet global standard Array (lui-même héritant de Object).
L'objet Array représente une liste d'éléments indexés, permettant de stocker et de récupérer des données facilement.

Déclaration d'un tableau vide

```
// Déclaration explicite en instanciant l'objet Array
let tableau1 = new Array();
// Création d'un tableau avec une taille de 3 (non
initialisé)
let tableau2 = new Array(3);
// Déclaration d'un tableau vide de manière littérale
let tableau3 = [];
```

Déclaration et initialisation

Déclaration d'un tableau vide

```
// Déclaration d'un tableau avec des valeurs initiales
let tableau1 = new Array(5, 10, 15, 20);
// Déclaration d'un tableau de manière littérale avec des
chaînes de caractères
let tableau2 = ['A', 'B', 'C'];
// Affichage des tableaux
console.log(tableau1); // Affiche : [5, 10, 15, 20]
console.log(tableau2); // Affiche : ['A', 'B', 'C']
```

Taille du tableau : La propriété length de l'objet Array retourne la taille du tableau

Déclaration d'un tableau vide

```
// Déclaration d'un tableau de manière littérale avec des
chaînes de caractères
let tableau = ['A', 'B', 'C'];
// Calcul du nombre d'éléments dans le tableau
let nbElements = tableau.length;
// Affichage du nombre d'éléments
console.log(nbElements); // Affiche : 3
```

Accéder aux éléments du tableau

Pour accéder à un élément d'un tableau en JavaScript, on utilise l'indice de l'élément souhaité. L'indice est un nombre qui indique la position de l'élément dans le tableau.

Accéder à un élément du tableau

```
// Déclaration d'un tableau avec des éléments
let tableau = ['A', 'B', 'C'];
// Accéder à l'élément à l'indice 1 (le deuxième élément)
console.log(tableau[1]); // Affiche : B
```

Récupérer l'indice d'un élément la méthode indexOf()

Récupérer l'indice d'un élément

```
// Déclaration d'un tableau avec des éléments
let tableau = ['A', 'B', 'C'];
// Utilisation de la méthode indexOf pour trouver l'indice
de l'élément 'C'
console.log(tableau.indexOf('C')); // Retourne 2
```

NB : indexOf a un deuxième paramètre optionnel qui permet aussi de choisir l'indice à partir duquel la recherche débute (Par défaut, ce deuxième paramètre est à 0).
Exemple : console.log (tableau.indexOf('B', 2)) ;

Parcourir un tableau

On peut parcourir un tableau de différente manières :

Récupérer l'indice d'un élément

```
for(let i = 0; i < monTableau.length ; i ++)
{
// monTableau [i] permet d'accéder à l'élément courant du
tableau
}
//Ou bien
for (const monElement of monTableau )
{
// monElement permet d'accéder à l'élément courant du
tableau
}
```

Parcourir un tableau

Récupérer l'indice d'un élément

```
monTableau.forEach (monElement => {
// monElement permet d'accéder à l'élément courant du
tableau
});
// Ou bien
monTableau.forEach ( function (monElement)
{
// monElement permet d'accéder à l'élément courant du
tableau
});
```

Parcourir un tableau

Avec une fonction fléchée

```
let monTableau = ['A', 'B', 'C'];
// Utilisation de forEach avec une
fonction fléchée
monTableau.forEach((monElement) => {
console.log(monElement); // Affiche
chaque élément du tableau
});
```

Avec une fonction anonyme

```
let monTableau = ['A', 'B', 'C'];
// Utilisation de forEach avec une
fonction anonyme
monTableau.forEach(function(monElement)
{
console.log(monElement); // Affiche
chaque élément du tableau
});
```

Ajouter un élément dans un tableau

La méthode push ajoute un élément à la fin du tableau

Ajouter un élément dans un tableau

```
let tableau = ['A', 'B']; // Déclaration du tableau avec
des éléments 'A' et 'B'
// Utilisation de la méthode ‘push‘ pour ajouter un nouvel
élément 'C' à la fin du tableau
tableau.push('C');
// Affichage du contenu du tableau
console.log(tableau); // Retourne : ['A', 'B', 'C']
```

Ajouter un élément dans un tableau

La méthode unshift ajoute un élément au début du tableau

Ajouter un élément dans un tableau

```
let tableau = ['A', 'B']; // Déclaration du tableau avec
les éléments 'A' et 'B'
// Utilisation de la méthode ‘unshift‘ pour
ajouter un nouvel élément '22' au début du tableau
tableau.unshift(22);
// Affichage du contenu du tableau
console.log(tableau); // Retourne [22, 'A', 'B']
```

Modifier un élément du tableau

Pour modifier la valeur d'un élément, on peut directement utiliser son indice

Modifier un élément du tableau

```
let tableau = ['B', 'B', 'C']; // Déclaration du tableau
initial avec les éléments 'B', 'B' et 'C'
// Modifie le premier élément du tableau (indice 0)
tableau[0] = 'A';
// Ajoute un élément 'E' à l'indice 4
tableau[4] = 'E';
// Affichage du tableau
console.log(tableau); // Retourne ['A', 'B', 'C', <empty>,
'E']
```

Supprimer un élément du tableau

La méthode pop() supprime le dernier élément du tableau

Supprime le dernier élément du tableau

```
let tableau = ['A', 'B', 'C']; // Déclaration initiale du
tableau
tableau.pop(); // Supprime le dernier élément du tableau
console.log(tableau); // Affiche ['A', 'B']
```

Supprimer un élément du tableau

La méthode shift() supprime le premier élément du tableau

Supprime le premier élément du tableau

```
let tableau = ['A', 'B', 'C']; // Déclaration initiale du tableau
tableau.shift(); // Supprime le premier élément du tableau
console.log(tableau); // Affiche ['B', 'C']
```

Supprimer un élément du tableau

La méthode splice permet de supprimer plusieurs éléments.
Le premier paramètre est l'indice à partir duquel supprimer, et le second est le nombre d'éléments à supprimer.

Supprime le premier élément du tableau

```
let tableau = ['A', 'B', 'C']; // Déclaration initiale du tableau
tableau.splice(1, 1); // Supprime l'élément à l'indice 1 (ici 'B')
console.log(tableau); // Affiche ['A', 'C']
```

Trier un tableau

La méthode sort() retourne les éléments par ordre alphabétique (elle doivent être de la même nature).

Trier un tableau

```
let tableau = ['E', 'A', 'D']; // Déclaration initiale du
tableau
tableau.sort(); // Trie le tableau par ordre alphabétique
console.log(tableau); // Affiche ['A', 'D', 'E']
```

Inverser un tableau

La méthode reverse() Inverse l'ordre des éléments (sans tri).

Inverser un tableau

```
let tableau = ['A', 'B', 'C']; // Déclaration initiale du
tableau
tableau.reverse(); // Inverse l'ordre des éléments du
tableau
console.log(tableau); // Affiche ['C', 'B', 'A']
```

Recherche un élément dans le tableau

La méthode findIndex() retourne l'indice du premier élément du tableau qui remplit une condition

Recherche un élément dans le tableau

```
function findC(element) {
return element === 'C'; // Vérifie si l'élément est égal à
'C'
}
let tableau = ['A', 'B', 'C', 'D', 'C'];
let index = tableau.findIndex(findC); // Retourne 2,
l'indice du premier 'C'
console.log(index); // Affiche 2
```

L'objet String (chaînes de caractères)

L'objet String de JavaScript est utilisé pour manipuler les chaînes de caractères.
Il possède de nombreuses méthodes.

Méthode	Description
`length`	C'est un entier qui indique la taille de la chaîne de caractères.
`charAt`	Méthode qui permet d'accéder à un caractère isolé d'une chaîne.
`substring(x, y)`	Méthode qui renvoie un string partiel situé entre la position x et la position y.
`toLowerCase()`	Transforme toutes les lettres en minuscules.
`toUpperCase()`	Transforme toutes les lettres en majuscules.

Table 6 – Méthodes de manipulation de chaînes de caractères

-Chercher dans une Chaine indexOf startsWith endsWith split.
-Transformer une chaine en tableau la méthode Array from permet de transformer une chaîne en un véritable tableau.

L'objet Date

L'objet date de JavaScript est utilisé pour obtenir la date (mois et jour) ainsi que le temps (minutes et secondes).
On peut utiliser 4 variantes du constructeur Date pour créer un objet date.
Constructeurs de l'objet Date :

- new Date() : Crée un objet Date avec la date et l'heure actuelles.
- new Date(dateString) : Crée un objet Date à partir d'une chaîne de caractères représentant une date valide.
- new Date(milliseconds) : Crée un objet Date correspondant à un nombre de millisecondes depuis le 1er janvier 1970 (heure UTC).
- new Date(year, monthIndex, day, hours, minutes, seconds, milliseconds) : Crée un objet Date avec une date et une heure spécifiques. Remarque : le mois est basé sur un index (0 =

Exemple

Inverser un tableau

```
let now = new Date(); // Date et heure actuelles
let birthday = new Date('1990-05-15'); // Date
d'anniversaire
let specificDate = new Date(2024, 0, 1); // 1er janvier
2024
```

L'objet Window

L'objet window de JavaScript est le parent de chaque objet qui compose la page web Il possède plusieurs méthodes. La méthode alert bloque le programme tant que l'utilisateur n'aura pas cliquer sur "OK" Utile pour le débugger les scripts.

Syntaxe

```
let variable = "Bonjour, "; // Déclare une variable
alerte("variable");
alert("chaîne de caractères"); // Affiche une boîte de
dialogue avec le texte "chaîne de caractères"
alert(variable + "chaîne de caractères"); // Affiche
"Bonjour, chaîne de caractères"
```

L'objet Window

La méthode confirm(texte) : permet d'avertir l'utilisateur en ouvrant une boite de dialogue avec deux choix "OK" et "Annuler" Le clique sur OK renvoie la valeur true.

Syntaxe

```
if (confirm('Voulez-vous continuer ?')) {
// action à faire pour la valeur true (si l'utilisateur
clique sur "OK")
console.log("L'utilisateur a cliqué sur OK.");
}else {
// action à faire pour la valeur false (si l'utilisateur
clique sur "Annuler")
console.log("L'utilisateur a cliqué sur Annuler.");
}
```

L'objet Window

La méthode prompt (boite d'invite) propose un champ comportant une entrée à compléter par l'utilisateur Cette entrée possède aussi une valeur par défaut En cliquant sur OK, la méthode renvoie la valeur tapée par l'utilisateur ou la réponse proposée par défaut Si l'utilisateur clique sur Annuler ou Cancel, la valeur null est alors renvoyée.

Syntaxe

```
prompt("texte de la boîte d'invite", "valeur par défaut");
```

Exemple

Exemple

```
let age = prompt("Quel est votre âge ?", "18");
if (age !== null) {
console.log("Votre âge est : " + age);
}else {
console.log("Vous avez annulé l'entrée.");
}
```

Sélecteurs CSS

En javaScript on peut chercher les éléments par leur sélecteur CSS

- La méthode querySelector() renvoie le premier élément qui correspond à un ou plusieurs sélecteurs CSS spécifiés.
- La méthode querySelectorAll() renvoie tous les éléments correspondants.

Exemple

Exemple

```
document.querySelector (sélecteur CSS)
document.querySelectorAll (sélecteur CSS)
```

La méthode querySelector()

La méthode querySelector()

```
// Sélectionne le premier élément <p> trouvé dans le document
const paragraphe = document.querySelector("p");
console.log(paragraphe);
// Sélectionne le premier élément <p> avec la classe "par"
const paragraphePar = document.querySelector("p.par");
console.log(paragraphePar);
// Modifie le contenu HTML de l'élément avec l'id "id1"
document.querySelector("#id1").innerHTML = "Bonjour!";
// Sélectionne le premier élément <p> qui est un enfant direct d'un
<div>
const paragrapheDansDiv = document.querySelector("div > p");
console.log(paragrapheDansDiv);
// Sélectionne le premier élément <a> qui a un attribut "target"
const lienAvecTarget = document.querySelector("a[target]");
console.log(lienAvecTarget);
```

Conclusion

La méthode querySelector() est un outil puissant pour manipuler le DOM en JavaScript. Elle est facile à utiliser pour sélectionner et modifier des éléments spécifiques sur une page web. Elle remplace souvent d'anciennes méthodes comme getElementById(), getElementsByClassName(), et getElementsByTagName(), grâce à sa capacité à utiliser des sélecteurs CSS complexes.

La méthode querySelectorAll()

Exemples d'utilisation de querySelectorAll()

```
// Sélectionne tous les éléments <p> dans le document
let x = document.querySelectorAll("p");
// Modifie la couleur d'arrière-plan du premier élément <p> (index 0)
x[0].style.backgroundColor = "red";
// Sélectionne tous les éléments <p> avec la classe "par"
let x = document.querySelectorAll("p.par");
// Modifie la couleur d'arrière-plan du premier élément <p> avec la classe "par"
x[0].style.backgroundColor = "red";
// Sélectionne tous les éléments avec la classe "par" et compte leur nombre
let x = document.querySelectorAll(".par");
let nombreElements = x.length;
console.log(nombreElements); // Affiche le nombre d'éléments avec la classe "par"
// Sélectionne tous les éléments avec la classe "par"
let x = document.querySelectorAll(".par");
// Parcourt tous les éléments sélectionnés et modifie leur couleur d'arrière-plan
for (let i = 0; i < x.length; i++) {
x[i].style.backgroundColor = "red"; }
// Sélectionne tous les éléments <a> avec l'attribut "target"
let x = document.querySelectorAll("a[target]");
// Parcourt tous les éléments sélectionnés et modifie leur bordure
for (let i = 0; i < x.length; i++) {
x[i].style.border = "2px solid red"; }
```

La méthode querySelectorAll()

Exemples d'utilisation de querySelectorAll()

```
// Sélectionner le premier paragraphe du document
document.querySelector("p").textContent = "1er paragraphe du document";
// Sélectionner le premier <div> du document
let documentDiv = document.querySelector("div");
// Modifier le texte du premier paragraphe à l'intérieur du premier
<div>
documentDiv.querySelector("p").textContent = "1er paragraphe du premier
div";
// Sélectionner le premier paragraphe avec la classe 'bleu' et changer
sa couleur en bleu
document.querySelector("p.bleu").style.color = "blue";
// Sélectionner tous les paragraphes à l'intérieur du premier <div>
let divParas = documentDiv.querySelectorAll("p");
```

Accéder à un élément du DOM

On peut Chercher les éléments directement par nom en utilisant les méthodes suivantes :

- document.getElementsByTagName()
- document.getElementsByClassName()
- document.getElementById()
- document.getElementsByName()

Ou bien en utilisant un sélecteur CSS associé

- document.querySelector()
- document.querySelectorAll()

Modes d'Accès aux éléments

Accéder à un élément en fonction de la valeur de son attribut id.

La méthode getElementById() renvoie un objet qui représente l'élément dont la valeur de l'attribut id correspond à la valeur spécifiée en argument.

Exemple

```
//Sélectionner l'élément avec un id = 'p1' et modifie la
couleur du texte
document.getElementById ('p1').style.color = 'blue';
```

Modes d'Accès aux éléments

Accéder à un élément en fonction de la valeur de son attribut class
La méthode getElementsByClassName() renvoie une liste des éléments possédant un attribut class avec la valeur spécifiée en argument.

Exemple

```
<!- HTML ->
<p class="bleu">Texte en bleu 1</p>
<p class="bleu">Texte en bleu 2</p>
<p>Texte normal</p>
<!- JavaScript ->
<script>
// Sélectionner les éléments avec la classe 'bleu'
let bleuElements = document.getElementsByClassName('bleu');
// Parcourir chaque élément et changer la couleur du texte
for (let element of bleuElements) {
element.style.color = 'blue';
}
</script>
```

Modes d'Accès aux éléments

Accéder à un élément en fonction de son identité (Nom de la balise).
La méthode getElementsByTagName() permet de sélectionner des éléments en fonction de leur nom.

Exemple

```
<!- HTML ->
<p>Paragraphe 1</p>
<p>Paragraphe 2</p>
<p>Paragraphe 3</p>
<!- JavaScript ->
<script>
// Sélectionner tous les éléments <p> du document
let paras = document.getElementsByTagName('p');
// Parcourir chaque élément <p> et changer la couleur du texte en bleu
for (let element of paras) {
element.style.color = 'blue';
}
</script>
```

Modes d'Accès aux éléments

Accéder directement à des éléments particuliers avec les propriétés de Document. L'API DOM fournit également des propriétés permettant d'accéder directement à certains éléments du document Parmi ces propriétés on trouve.

- La propriété body qui retourne le noeud représentant l'élément body ;
- La propriété head qui retourne le noeud représentant l'élément head
- La propriété links qui retourne une liste de tous les éléments « a » ou « area » possédant un attribut href avec une valeur ;
- La propriété title qui retourne le titre (le contenu de l'élément title) du document
- La propriété url qui renvoie l'URL du document sous forme d'une chaine de caractères ;
- La propriété scripts qui retourne une liste de tous les éléments script du document ;
- La propriété cookie qui retourne la liste de tous les cookies associés au document sous forme d'une chaine de caractères.

Exemple

Exemple

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Ancien Titre</title>
</head>
<body>
<h1>Bienvenue sur la page</h1>
<script>
// Sélectionner l'élément <body> et appliquer une couleur bleue
document.body.style.color = 'blue';
// Modifier le texte de l'élément <title>
document.title = 'Le DOM';
</script>
</body>
```

Créer un élément en JavaScript

La méthode createElement permet de créer de nouveaux éléments dans le document.
La variable element renvoie la référence de l'élément crée.

Remarque

`L'élément crée par la méthode createElement() ne s'attache pas automatiquement au document.`

Exemple

Exemple

```
let element1 = document.createElement('p');
console.log(element1); // <p></p>
let element2 = document.createElement('div');
console.log(element2); // <div></div>
let element3 = document.createElement('DIV');
console.log(element3); // <div></div>
```

Créer un élément en JavaScript

La méthode createElement() de JavaScript crée un nouvel élément HTML, mais cet élément n'est pas ajouté automatiquement au DOM (Document Object Model). Cela signifie que même si vous créez un élément, il reste en mémoire, et vous devez spécifier explicitement où vous voulez l'ajouter dans la page.

Ajouter un élément en JavaScript

Pour ajouter un élément à l'arborescence du DOM (après l'avoir crée), il faut l'attacher à un élément parent La méthode append() insère un objet en tant que dernier enfant d'un élément parent.

Exemple

Exemple

```
<body>
<div id="parent">
<h1>Bienvenue</h1>
</div>
<script>
// Sélectionner l'élément parent par son ID
let parent = document.getElementById('parent');
// Créer un nouvel élément <p> (élément enfant)
let enfant = document.createElement('p');
// Ajouter du texte à l'élément enfant
enfant.innerHTML = "C'est un nouveau élément";
// Attacher l'élément enfant à l'élément parent
parent.append(enfant);
</script>
</body>
```

Supprimer un élément en JavaScript

La méthode removeChild supprime un élément de la structure du DOM. Le noeud à supprimer est passé en argument à la méthode. Une référence vers le noeud supprimé est retournée à la fin.

Exemple

```
let parent = document.getElementById('parent'); // Sélectionner un
élément parent
let enfant = document.getElementById('eltSupp'); // Sélectionner un
élément enfant
parent.removeChild(enfant); // Supprimer l'élément enfant
```

Exemple

Exemple

```
<body>
<!- Parent div ->
<div id="parent">
<p id="eltSupp">Ceci est un paragraphe à supprimer.</p>
<p>Ceci est un autre paragraphe.</p>
</div>
<button onclick="supprimerElement()">Supprimer le paragraphe</button>
<script>
function supprimerElement() {
// Sélectionner l'élément parent
let parent = document.getElementById('parent');
// Sélectionner l'élément enfant (celui à supprimer)
let enfant = document.getElementById('eltSupp');
// Supprimer l'élément enfant du parent
parent.removeChild(enfant);
}
</script>
</body>
```

Modifier un élément en JavaScript

La méthode replaceChild remplace un noeud par un autre noeud dans le DOM Une référence vers le noeud remplacé est retournée à la fin.

Syntaxe

```
parent.replaceChild (nouveauElement, ancienElement);
```

Modifier un élément en JavaScript

La méthode replaceChild remplace un noeud par un autre noeud dans le DOM Une référence vers le noeud remplacé est retournée à la fin.

Syntaxe

```
let parent = document.getElementById('parent'); //
Sélectionner un élément parent
let ancienElement = document.getElementById('id1'); //
Sélectionner l'ancien élément avec l'ID 'id1'
let nouvElement = document.createElement('h2'); // Créer
un nouveau élément de type <h2>
nouvElement.innerHTML = 'C'est le nouveau élément.'; //
Ajouter du texte à ce nouveau <h2>
parent.replaceChild(nouvElement, ancienElement); //
Remplacer l'ancien élément par le nouveau dans le parent.
```

Mettre à jour le style

Exemple

```
<div>
<div>
<label for="b1">Nom :</label>
<br>
<input type="text" class="style1" id="b1">
</div>
</div>
<script>
// Modifier le style de l'élément qui a l'attribut class="style1"
let element = document.getElementsByClassName('style1')[0]; //
Sélectionner le premier élément avec la classe 'style1'
element.style.borderColor = 'red'; // Modifier la couleur de la bordure
</script>
```

Définir le style à l'aide de element.className

La propriété element.className permet de changer les paramètres de style d'un élément HTML en lui attribuant une nouvelle classe dont le nom est passé à l'élément sélectionné.

Définir le style à l'aide de element.className

Exemple

```
<div>
<div>
<label for="b1">Nom :</label>
<br>
<input type="text" class="style1" id="b1">
</div>
</div>
<script>
// Sélectionner l'élément avec la classe 'style1'
let elements = document.getElementsByClassName('style1');
// Parcourir tous les éléments avec la classe 'style1' et leur ajouter
la classe 'styleErreur'
for (let element of elements) {
element.classList.add('styleErreur');
}
</script>
```

Mise à jour d'un attribut avec setAttribute

La méthode setAttribute() est utilisée pour définir un attribut à l'élément spécifié Si l'attribut existe déjà, sa valeur est mise à jour Sinon, un nouvel attribut est ajouté avec le nom et la valeur spécifiés.

Exemple

Exemple

```
<button type="button" id="Btn">Click button</button>
<script>
// Sélectionner l'élément avec l'ID 'Btn'
let btn = document.getElementById('Btn');
// Ajouter les attributs
btn.setAttribute('class', 'style1'); // Ajouter la classe 'style1'
btn.setAttribute('disabled', ''); // Désactiver le bouton en ajoutant
l'attribut 'disabled'
</script>
```

Suppression d'attributs d'éléments

La méthode removeAttribute() est utilisée pour supprimer un attribut d'un élément spécifié.

Exemple

```
<a href="https://www.ofppt.com/" id="lien">OFPPT</a>
<script>
// Sélectionner l'élément avec l'ID 'lien'
let lien = document.getElementById('lien');
// Supprimer la valeur de l'attribut 'href'
lien.removeAttribute('href');
</script>
```