Introduction

Introduction au langage Javascript

-

Introduction

-
-
-
-
-
-

Les concepts de base de JavaScript

-
-
-
-
-

-

Les structures conditionnelles

-
-
-
-

-

Les structures itératives

-
-
-
-
-
-

-

Les fonctions

-
-
-
-

-

L'exécution d'un code JS

-
- Résumé et transmission

JavaScript est un langage de programmation largement utilisé principalement pour le développement web, bien qu'il puisse également être utilisé dans d'autres contextes. Voici quelques points clés à retenir à propos de JavaScript

Définition de JavaScript :

JavaScript est un langage de programmation de haut niveau, interprété et orienté objet. Il a été initialement créé pour être utilisé dans les navigateurs web pour rendre les pages interactives et dynamiques.

Utilités de JavaScript :

- **Interactivité web** : L'une des utilisations les plus courantes de JavaScript est d'ajouter de l'interactivité aux sites web. Il permet de créer des fonctionnalités dynamiques telles que des formulaires interactifs, des animations, des menus déroulants et des mises à jour de contenu en temps réel sans avoir à recharger la page.

- **Manipulation du DOM** : JavaScript peut être utilisé pour manipuler le Document Object Model (DOM), c'est-à-dire la structure représentant les éléments d'une page web. Cela permet de modifier, ajouter ou supprimer des éléments de la page de manière dynamique.
- **Validation de formulaires** : JavaScript peut être utilisé pour valider les données entrées dans les formulaires avant qu'elles ne soient soumises au serveur, assurant ainsi une meilleure expérience utilisateur et une gestion plus propre des données.
- **Gestion d'événements** : JavaScript permet de gérer les interactions de l'utilisateur avec la page en répondant à des événements tels que les clics de souris, les pressions de touches, les mouvements de souris, etc.
- **Applications web** : Avec l'avènement des frameworks comme React, Angular et Vue.js, JavaScript est également utilisé pour créer des applications web complexes et à grande échelle, allant au-delà des fonctionnalités de base des sites web.
- **Programmation côté serveur** : Avec l'utilisation de Node.js, JavaScript peut également être utilisé pour la programmation côté serveur, ce qui permet de construire des applications web côté serveur avec le même langage.
- **Interfaces utilisateur** : Les bibliothèques et les frameworks JavaScript permettent de créer des interfaces utilisateur élégantes et réactives, améliorant ainsi l'expérience globale de l'utilisateur.

Globalement, JavaScript est devenu un élément essentiel du développement web moderne, jouant un rôle clé dans la création d'expériences utilisateur riches et interactives.

1 / 7

Introduction

2 / 7

Les concepts de base de JavaScript

Introduction au langage Javascript

-

Introduction

Les concepts de base de JavaScript

Les structures conditionnelles

Les structures itératives

-
-

-

Les fonctions

-
-
-
-

-

L'exécution d'un code JS

-
- Résumé et transmission

Les variables

Déclaration des variables

Il existe trois manières pour déclarer une variable dans Javascript. Noter tout d'abord que Javascript est sensible à la casse.

- **var :** On déclare une variable, éventuellement en initialisant sa valeur.
- **let** : On déclare une variable dont la portée est celle du bloc courant, éventuellement en initialisant sa valeur.
- **Const** : On déclare une constante nommée, dont la portée est celle du bloc courant, accessible en lecture seule.

Exemple :

```
var a = 10;
```

```
let monVariable =  '' Bob' ' ;
```

```
const pi=3.14;
```

Portée de la variable

Avec `var`, la portée de la variable est globale ou limitée à la fonction dans laquelle elle est déclarée. Cela signifie que la variable est accessible de partout dans la fonction ou même en dehors de la fonction si elle est déclarée en dehors de toutes les fonctions.

Avec `let`, la portée de la variable est limitée au bloc (généralement entre des accolades `{}`) dans lequel elle est déclarée. Cela signifie que la variable ne sera accessible qu'à l'intérieur de ce bloc.

Exemple :

```
if (true) {

  var x = 5;

}

console.log(x); // x vaut 5
```

Exemple :

```
if (true) {

  let y = 5;

}

console.log(y); // ReferenceError: y is not defined
```

Hoisting

Les déclarations `var` sont soumises à une étape appelée "hoisting" (élévation). Cela signifie que même si vous déclarez une variable plus tard dans le code, elle sera "élevée" en haut de sa portée actuelle lors de l'exécution. Cela peut parfois entraîner des comportements inattendus.

Les déclarations `let` ne sont pas soumises à un hoisting aussi agressif. Elles sont également "élevées", mais elles ne sont pas initialisées avant leur déclaration réelle, ce qui peut éviter certaines surprises.

Exemple :

```
console.log(x === undefined); // donne "true"
```

```
var x = 3;
```

En règle générale, il est recommandé d'utiliser `let` (et `const` lorsque la valeur ne changera pas) plutôt que `var` dans le développement JavaScript moderne. `let` offre un meilleur contrôle de la portée, évite certaines erreurs courantes et est plus conforme aux bonnes pratiques en matière de programmation.

Types de données

Les variables peuvent contenir des différents types de données :

Type	Exemple et Explication
null	let monVariable = null /* null est différent de Null et de NULL */
undefined	let monVariable = undefined /* utilisé généralement dans les tests conditionnels */
Chaine de caractère (String)	let monVariable = '' hello''
Nombre	let monVariable = 5
Booléen	let monVariable = true
Tableau	let monVariable = ['' hello'' , 5, 10, '' good''] /* compteur commence de 0 */
Object	let monVariable = document.querySelector('h1');

Exemple :

```
var message = 42 ;
```

```
message= "hello world"; // pas d'erreur ici
```

Exemple :

```
x = "La réponse est " + 42; // "La réponse est 42"
```

```
y = 42 + " est la réponse"; // "42 est la réponse"
```

Exemple :

```
"37" - 7; // 30
```

```
"37" + 7; // "377"
```

Exemple : Conversion de types :

```
/* le premier paramètre est la chaine de caractère, le deuxième
```

```
paramètre est la base, ici la base 2 */
```

```
parseInt("101", 2); // 5
```

```
/* autre manière pour convertir la chaine de caractère en nombre */
```

```
+"1.1" = 1.1; // fonctionne seulement avec le + unaire
```

Les littéraux

Un littéral est une notation utilisée pour représenter directement des valeurs de types différents, tels que des chaînes de caractères, des nombres, des tableaux, des objets, des expressions régulières, etc. Les littéraux sont une manière concise et directe de définir des valeurs sans avoir à passer par des expressions ou des constructions plus complexes.

Les littéraux sont très utiles pour définir des valeurs de manière claire et concise, et ils sont largement utilisés dans le code JavaScript pour initialiser des variables, des propriétés d'objets, des éléments de tableau, etc. Ils font partie intégrante de la syntaxe du langage et permettent d'écrire du code de manière plus lisible et plus expressive.

Littéral de chaine de caractère :

Voici des exemples :

```
var message = "Bonjour, monde!";
```

```
var autreMessage = "Une ligne \n une autre ligne" ;
```

```
var nom = "Robert",
```

```
          jour = "aujourd'hui";
```

```
         `Bonjour ${nom}, comment allez-vous ${jour} ?`;
```

```
var citation = "Il lit \"Bug Jargal\" de V. Hugo."; //Il lit "Bug
```

```
Jargal" de V. Hugo
```

Les caractères et ses significations

- \0 : Octet null
- \b : Retour arrière
- \f : Saut de page
- \n : Nouvelle ligne
- \r : Retour chariot
- \t : Tabulation
- \v : Tabulation verticale
- \' : Apostrophe ou guillemet droit simple
- \" : Guillemet droit double

Littéral de nombre

```
var age = 25;  // entier
```

```
var prix = 19.99; //décimal
```

Littéral de tableaux :

```
var fruits = ["pomme", "banane", "orange"];
```

```
var couleur = ["noir ", , "rouge"] ; // couleur[1] est undefined
```

Littéral d'objet :

```
var personne = {
    nom: "Doe",
    prénom: "John",
    âge: 30
};
```

Littéral d'expression régulière

```
var regex = /[a-zA-Z]+/;
```

Littéral de booléen

```
var estVrai = true;
var estFaux = false;
```

Littéral null

```
var valeurNulle = null;
```

Littéral undefined

```
var valeurNonDéfinie = undefined;
```

Les opérateurs

Les opérateurs sont des symboles spéciaux dans le langage de programmation JavaScript qui permettent d'effectuer des opérations sur des valeurs. Ils agissent sur les opérandes (les valeurs) et produisent un résultat. JavaScript propose différents types d'opérateurs pour effectuer des calculs, des comparaisons, des opérations logiques, etc. Voici une explication des principaux types d'opérateurs en JavaScript :

1.

Opérateurs Arithmétiques :

2.

- `+` (Addition) : Ajoute deux valeurs numériques.
- `-` (Soustraction) : Soustrait la deuxième valeur de la première.
- `*` (Multiplication) : Multiplie deux valeurs numériques.
- `/` (Division) : Divise la première valeur par la deuxième.
- `%` (Modulo) : Donne le reste de la division de la première valeur par la deuxième.
- `**` (Exponentiation) : Élève la première valeur à la puissance de la deuxième.

```
let a = 10;
```

```
let b = 3;
```

```
let addition = a + b; // 10 + 3 = 13
```

```
let soustraction = a - b; // 10 - 3 = 7
```

```
let multiplication = a * b; // 10 * 3 = 30
```

```
let division = a / b; // 10 / 3 ≈ 3.333...
```

```
let modulo = a % b; // 10 % 3 = 1 (reste de la division)
```

```
let exponentiation = a ** b; // 10^3 = 1000
```

1.

Opérateurs de Comparaison :

2.

- `==` (Égal à) : Vérifie si deux valeurs sont égales en valeur (mais ne prend pas en compte le type).

- `==` (Égal à) : Vérifie si deux valeurs sont égales en valeur et en type.
- `!=` (Différent de) : Vérifie si deux valeurs sont différentes en valeur.
- `!==` (Différent de) : Vérifie si deux valeurs sont différentes en valeur ou en type.
- `<` (Inférieur à), `>` (Supérieur à), `<=` (Inférieur ou égal à), `>=` (Supérieur ou égal à) : Comparaison numérique entre deux valeurs.

```
let x = 5;
```

```
let y = "5";
```

```
console.log(x == y); // true (les valeurs sont égales)
```

```
console.log(x === y); // false (les types sont différents)
```

```
console.log(x != y); // false (les valeurs sont égales)
```

```
console.log(x !== y); // true (les types sont différents)
```

```
console.log(x < 10); // true
```

```
console.log(x >= 3); // true
```

1.

Opérateurs Logiques :

2.
- `&&` (ET logique) : Renvoie `true` si les deux opérandes sont `true`.
- `||` (OU logique) : Renvoie `true` si au moins l'un des opérandes est `true`.
- `!` (NON logique) : Inverse la valeur d'un booléen (de `true` à `false` ou vice versa).

```
let p = true;
```

```
let q = false;
```

```
console.log(p && q); // false (ET logique)
```

```
console.log(p || q); // true (OU logique)
```

```
console.log(!p); // false (NON logique)
```

1.

Opérateurs d'Assignation :

2.

- `=` (Affectation simple) : Affecte la valeur de droite à la variable de gauche.
- `+=`, `-=`, `*=`, `/=`, `%=`, `**=` : Effectuent une opération et assignent le résultat à la variable de gauche.

```
let num = 5;
```

```
num += 2; // num = num + 2 => num = 7
```

```
num -= 3; // num = num - 3 => num = 4
```

```
num *= 2; // num = num * 2 => num = 8
```

```
num /= 4; // num = num / 4 => num = 2
```

```
num %= 2; // num = num % 2 => num = 0 (reste de la division)
```

1.

Opérateurs d'Incrémentation et de Décrémentation :

2.

- `++` (Incrémentation) : Augmente la valeur d'une variable de 1.
- `--` (Décrémentation) : Diminue la valeur d'une variable de 1.

```
let counter = 10;
```

```
counter++; // counter = counter + 1 => counter = 11
```

```
counter--; // counter = counter - 1 => counter = 1
```

1.

Opérateurs Ternaires :

2.
- `condition ? valeurSiVraie : valeurSiFausse` : Effectue une opération conditionnelle. Si la condition est vraie, la première valeur est renvoyée ; sinon, c'est la deuxième valeur qui est renvoyée.

```
let age = 18;
```

```
let message = (age >= 18) ? "Majeur" : "Mineur";
```

```
console.log(message); // "Majeur"
```

1.

Opérateurs de Concaténation :

2.
- `+` (Concaténation) : Combine deux chaînes de caractères en une seule.

```
let firstName = "John";
```

```
let lastName = "Doe";
```

```
let fullName = firstName + " " + lastName;
```

```
console.log(fullName); // "John Doe"
```

1.

Opérateurs Bit à Bit (Bitwise) :

2.

- `&`, `|`, `^`, `~`, `<<`, `>>` : Ces opérateurs effectuent des opérations bit à bit sur les valeurs numériques. Ils sont souvent utilisés dans des contextes de manipulation de bits.

```
let a = 5; // binaire: 0101
```

```
let b = 3; // binaire: 0011
```

```
let bitwiseAnd = a & b; // binaire: 0001 => décimal: 1 (AND bit à bit)
```

```
let bitwiseOr = a | b; // binaire: 0111 => décimal: 7 (OR bit à bit)
```

```
let bitwiseXor = a ^ b; // binaire: 0110 => décimal: 6 (XOR bit à bit)
```

```
let bitwiseNotA = ~a; // binaire: 1010 => décimal: -6 (NOT bit à bit)
```

```
let leftShift = a << 1; // binaire: 1010 => décimal: 10 (Décalage à gauche)
```

```
let rightShift = a >> 1; // binaire: 0010 => décimal: 2 (Décalage à droite)
```

Opération	Descriptin	Symbole	Exemple
Addition	Ajouter des nombres, Concaténation	+	3+2 ; //5 ‘’ bon’ ’ +’ ’ jour’ ’ ; //bonjour
Soustraction, Multiplication,		-,*,/	9-2 ; 4*2 ;

Division			9/2 ;
Assignation	Affectation une valeur à une variable	=	message = ‘’ hello’’
Egalité	Le résultat est booléen	===	message === ‘’ bonjour ‘’
Négation,	Le résultat est booléen	!,	let monVariable = 3; !(monVariable === 3); // false
N’égale pas		!==	let monVariable = 3; monVariable !== 3; // false

Qu'est-ce qu'une variable en JavaScript ?

- Une fonction pré-définie.
- Un conteneur pour stocker des données.
- Un type de données numérique.
- Une structure de contrôle.

Les concepts de base de JavaScript

Les structures conditionnelles

Introduction au langage Javascript

Introduction

Les concepts de base de JavaScript

Les structures conditionnelles

Les structures itératives

- **Les fonctions**
-
-
-
-

- **L'exécution d'un code JS**
-
- Résumé et transmission

Instruction if .. else

Les instructions if et else sont utilisées pour créer des conditions dans votre code, ce qui vous permet d'exécuter différents blocs de code en fonction de certaines conditions booléennes.

Si la condition est vraie, les instructions du bloc correspondant seront exécutées

Sinon les instructions du bloc else seront exécutés

Structure 1

```
if (condition) {
```

```
  instruction_1;
```

```
} else {
```

```
  instruction_2;
```

```
}
```

Structure 2

Dans cette deuxième structure, à chaque fois, on vérifie la condition pour exécuter le bloc qui le suit. Si toutes les conditions sont fausses, les instructions du bloc else seront exécutés.

```
if (condition_1) {

  instruction_1;

} else if (condition_2) {

  instruction_2;

} else if (condition_n) {

  instruction_n;

} else {

  dernière_instruction;

}
```

Exemple 1:

```
var age = 18;

if (age >= 18) {

    console.log("Vous êtes majeur.");

}
```

Exemple 2:

```
var note = 75;

if (note >= 90) {
```

```
    console.log("Très bien !");

} else if (note >= 70) {

    console.log("Bien !");

} else if (note >= 50) {

    console.log("Satisfaisant.");

} else {

    console.log("Insuffisant.");

}
```

Exemple 3 :

```
var age = 20;

var message = (age >= 18) ? "Majeur" : "Mineur";

console.log(message);
```

Notons bien que les valeurs suivantes utilisées dans les conditions sont évaluées à false :

- false
- undefined
- null
- 0
- NaN
- la chaîne de caractères vide ("")

Les autres valeurs y compris les objets sont équivalents à true.

Instruction Switch

La valeur de la variable choix est comparé avec chaque cas à l'intérieur du bloc switch.

Si la valeur correspond à l'une des valeurs case, le code à l'intérieur de ce case est exécuté. L'utilisation de l'instruction break est importante, car elle indique à JavaScript de sortir de la structure switch une fois qu'un cas correspondant a été trouvé. Si le break est omis, tous les cas suivants seront également exécutés, même s'ils ne correspondent pas.

Si aucun des cas ne correspond à la valeur de l'expression, le code à l'intérieur du default sera exécuté. Si aucune clause default n'est trouvée, le programme reprend l'exécution à l'instruction qui suit la fin de switch. Par convention, la clause default est écrite comme la dernière clause, mais il n'est pas nécessaire que ce soit le cas.

```
switch (choix) {

  case label_1:

    instructions_1

    [break;]


  case label_2:

    instructions_2

    [break;]


  ...


  default:

    instructions_par_defaut
```

```
    [break;]
```

```
}
```

Exemple :

```
var jourDeLaSemaine = "Lundi";
```

```
switch (jourDeLaSemaine) {
```

```
    case "Lundi":
```

```
        console.log("C'est le début de la semaine.");
```

```
        break;
```

```
    case "Mardi":
```

```
        console.log("C'est le deuxième jour de la semaine.");
```

```
        break;
```

```
    case "Mercredi":
```

```
        console.log("C'est le milieu de la semaine.");
```

```
        break;
```

```
    case "Jeudi":

        console.log("C'est le quatrième jour de la semaine.");

        break;


    case "Vendredi":

        console.log("C'est presque le week-end !");

        break;


    case "Samedi":


    case "Dimanche":

        console.log("C'est le week-end !");

        break;


    default:

        console.log("Jour non reconnu.");

}
```

Quelle sera la sortie de ce code ?

```
var age = 17;
if (age >= 18) {
    console.log("Majeur");
} else if (age >= 13 && age < 18) {
    console.log("Adolescent");
} else {
    console.log("Enfant");
}
```

- "Majeur"
-
- "Adolescent"
-
- "Enfant"
-
- Aucune des réponses
-

3 / 7

Les structures conditionnelles

4 / 7

Les structures itératives

-

-
-

Les concepts de base de JavaScript

Les structures conditionnelles

Les structures itératives

-

-

  Les fonctions

-

-
-
-

-

  L'exécution d'un code JS

-
- Résumé et transmission

L’ instruction for

```
for (initialisation; condition; itération) {
```

```
    // Bloc de code à exécuter à chaque itération
```

```
}
```

- L'initialisation est une expression qui est exécutée une seule fois avant le début de la boucle. C'est souvent utilisé pour déclarer des compteurs ou des variables de contrôle.
- La condition est une expression booléenne évaluée avant chaque itération. Si cette condition devient fausse, la boucle se termine.
- L'itération est une expression exécutée après chaque itération de la boucle. Elle est généralement utilisée pour mettre à jour les compteurs ou les variables de contrôle. Dans la boucle for, le nombre des itérations est fini.

Exemple : parcourir un tableau

```
var nombres = [1, 2, 3, 4, 5];
```

```
for (var i = 0; i < nombres.length; i++) {
```

```
    console.log(nombres[i]);
```

```
}
```

L’ instruction do.. while

```
do {
```

```
    // Bloc de code à exécuter
```

```
} while (condition);
```

Le bloc de code à l'intérieur des accolades est exécuté au moins une fois, indépendamment de la valeur de la condition.

Après l'exécution du bloc de code, la condition est évaluée. Si la condition est vraie, le bloc de code est réexécuté. Si la condition est fausse, la boucle se termine.

Exemple :

```
var compteur = 0;
```

```
do {
```

```
    console.log("Itération : " + compteur);
```

```
    compteur++;
```

```
} while (compteur < 5);
```

L’ instruction while

```
while (condition) {
```

```
    // Bloc de code à exécuter
```

```
}
```

La condition est une expression booléenne. Tant que cette condition est vraie, le bloc de code à l'intérieur des accolades sera exécuté.

Après chaque itération, la condition est évaluée à nouveau. Si la condition est toujours vraie, la boucle continue. Si la condition devient fausse, la boucle s'arrête et l'exécution se poursuit après le bloc while.

L'instruction while est particulièrement utile lorsque vous ne savez pas à l'avance combien d'itérations seront nécessaires, mais vous savez que vous devez répéter une action tant qu'une certaine condition est respectée. Cependant, assurez-vous que la condition est mise à jour correctement à l'intérieur de la boucle pour éviter les boucles infinies.

Exemple :

```
var compteur = 0;

while (compteur < 5) {

    console.log("Itération : " + compteur);

    compteur++;

}
```

Les instructions for .. in, for .. of

L'instruction for..of crée une boucle qui fonctionne avec les objets itérables (qui incluent Array, Map, Set, l'objet arguments, etc.). La boucle appelle un mécanisme d'itération propre à l'objet utilisé et elle parcourt l'objet et les valeurs de ses différentes propriétés

```
for (variable of objet) {

  instruction

}
```

L'instruction for...in permet de parcourir toutes les propriétés énumérables d'un objet, et pour chaque propriété, JavaScript exécutera la déclaration spécifiée.

Ainsi, si l'on modifie un objet Array en lui ajoutant des propriétés et/ou des méthodes, la boucle for...in renverra le nom de ces nouvelles propriétés en plus des indices des éléments du tableau.

```
for (variable in array) {

  instruction

}
```

Exemple important:

```
let arr = [3, 5, 7];

arr.toto = "coucou";



for (let i in arr) {

  console.log(i); // affiche 0, 1, 2, "toto" dans la console

}


for (let i of arr) {

  console.log(i); // affiche 3, 5, 7 dans la console

}
```

Quelle sera la sortie de ce code ?

```
for (var i = 1; i <= 5; i++) {
  if (i === 3) {
```

```
        continue;
    }
    console.log(i);
}
```

- 1 2 4 5
-
- 1 2 3 4
-
- 1 2 4
-
- 1 2 3 4 5
-

4 / 7

Les structures itératives

5 / 7

Les fonctions

-
-
-
-
-
-

Les structures conditionnelles

-
-
-
-
-

Les structures itératives

-
-
-
-
-
-

Les fonctions

-
-
-
-
-
-
-
-

- **L'exécution d'un code JS**
-
- Résumé et transmission

Définir des fonctions

Vous pouvez définir une fonction en utilisant le mot-clé function, suivi par un nom de fonction, des parenthèses contenant éventuellement des paramètres et un bloc de code entre des accolades. Voici la syntaxe générale pour définir une fonction en JavaScript :

```
function nomDeLaFonction(paramètre1, paramètre2, ...) {

    // Bloc de code à exécuter

}
```

- nomDeLaFonction est le nom que vous choisissez pour votre fonction.
- paramètre1, paramètre2, etc. sont les paramètres que la fonction peut accepter. Les paramètres sont optionnels.
- Le bloc de code entre les accolades {} est le code que la fonction exécutera lorsque vous l'appellerez.

Exemple 1 :

```
function direBonjour(nom) {

    console.log("Bonjour, " + nom + " !");

}
```

Vous pouvez également définir des fonctions anonymes (sans nom) et les stocker dans des variables :

Exemple 2 :

```
var addition = function(a, b) {
```

```
    return a + b;

};

console.log(addition(5, 3)); // Affiche 8

var x = addition(5,3);
```

Cependant, le nom de la fonction peut être utilisé dans le cas de l'utiliser dans la fonction elle-même, ce qu'on appelle la récursivité.

Exemple :

```
var factorielle = function fac(n) {

  return n < 2 ? 1 : n * fac(n - 1);

};
```

- nomDeLaFonction est le nom que vous choisissez pour votre fonction.
- paramètre1, paramètre2, etc. sont les paramètres que la fonction peut accepter. Les paramètres sont optionnels.
- Le bloc de code entre les accolades {} est le code que la fonction exécutera lorsque vous l'appellerez.

Exemple 1 :

```
function direBonjour(nom) {

    console.log("Bonjour, " + nom + " !");

}
```

Vous pouvez également définir des fonctions anonymes (sans nom) et les stocker dans des variables :

Exemple 2 :

```
var addition = function(a, b) {
    return a + b;
};
console.log(addition(5, 3)); // Affiche 8
var x = addition(5,3);
```

Cependant, le nom de la fonction peut être utilisé dans le cas de l'utiliser dans la fonction elle-même, ce qu'on appelle la récursivité.

Exemple :

```
var factorielle = function fac(n) {
  return n < 2 ? 1 : n * fac(n - 1);
};
```

Appeler des fonctions

```
direBonjour("Alice");
var x = addition(5,3);
console.log(factorielle(3));
```

Les fonctions doivent être situées dans la portée où elles sont invoquées, mais la déclaration d'une fonction peut être réalisée après l'appel seulement lorsque la fonction est définie avec la syntaxe function nomFonction(){}

Exemple important :

```
console.log(carré);     // La fonction carré est remontée/hoisted
mais vaut undefined

console.log(carré(5)); // TypeError: carré is not a function

var carré = function (n) {
  return n * n;
};
```

// Et avec let...

```
console.log(carré2);     // ReferenceError: carré2 is not defined

console.log(carré2(5)); // TypeError: carré2 is not a function

let carré2 = function (n) {
  return n * n;
};
```

Quelle est la manière correcte de définir une fonction en JavaScript ?

- function => maFonction { }
- function maFonction() { }
- function = maFonction() { }
- function maFonction { }

5 / 7

Les fonctions

6 / 7

L'exécution d'un code JS

Introduction au langage Javascript

- **Introduction**
- **Les concepts de base de JavaScript**
- **Les structures conditionnelles**

- **Les structures itératives**

- **Les fonctions**

- **L'exécution d'un code JS**

- Résumé et transmission

Deux méthodes pour exécuter un code JavaScript :

Navigateur Web :

1. Ouvrez un éditeur de texte (comme Visual Studio Code)
2. Ecrivez votre code JavaScript.
3. Enregistrez le fichier avec l'extension .js (par exemple, monscript.js).
4. Ouvrez un navigateur web (comme Chrome, Firefox, Safari, etc.).
5. Ouvrez la console de développement du navigateur (généralement en appuyant sur F12 ou Ctrl+Shift+I), puis naviguez vers l'onglet "Console".
6. Utilisez la balise <script> dans un fichier HTML pour lier votre script, ou saisissez directement votre code JavaScript dans la console et appuyez sur Entrée

```
<!DOCTYPE html>
```

```
<html>
<body> <script type="text/javascript"> ...</script> </body>
</html>
```

Node.js :

1. Télécharger depuis le site officiel de Node.js (https://nodejs.org/fr/download).
2. Installez Node.js sur votre ordinateur (si ce n'est pas déjà fait).
3. Ouvrez un éditeur de texte et écrivez votre code JavaScript.
4. Enregistrez le fichier avec l'extension .js.
5. Ouvrez un terminal ou une invite de commandes.
6. Utilisez la commande node nomdufichier.js pour exécuter votre script (par exemple, node monscript.js).

L'exécution d'un code JS