Formation Vue.js

Chapitre 2

JavaScript

Ibrahim ALAME

2025

Objectifs

Ce chapitre couvre les fondamentaux de JavaScript :

- Comprendre les bases du langage (variables, types, opérateurs)
- Maîtriser les structures de contrôle (conditions, boucles)
- Travailler avec les types complexes (objets, tableaux, fonctions)
- Manipuler le DOM pour créer des pages interactives
- Gérer les événements utilisateur
- Organiser son code avec les modules et les classes

Prérequis : Chapitre 1 - HTML et CSS
Durée estimée : 15 à 20 heures

Contents

1 Introduction

1.1 Qu'est-ce que JavaScript ?

Définition

JavaScript est un langage de programmation interprété, principalement utilisé pour rendre les pages web interactives. C'est le troisième pilier du développement web avec HTML et CSS.

1.1.1 Caractéristiques principales

- **Langage interprété** : exécuté directement par le navigateur sans compilation
- **Typage dynamique** : les types sont déterminés à l'exécution
- **Multi-paradigme** : supporte la programmation impérative, fonctionnelle et orientée objet
- **Orienté événements** : réagit aux actions de l'utilisateur
- **Mono-thread** : exécute une seule tâche à la fois (avec gestion asynchrone)

1.1.2 Cas d'utilisation

- **Frontend** : manipulation du DOM, animations, validation de formulaires
- **Backend** : avec Node.js pour créer des serveurs
- **Mobile** : avec React Native, Ionic, etc.
- **Desktop** : avec Electron (VS Code, Discord, Slack...)

1.2 Les versions de JavaScript et la compatibilité

JavaScript est standardisé sous le nom **ECMAScript** (ES). Les versions importantes :

Version	Année	Nouveautés majeures
ES5	2009	JSON, strict mode, forEach
ES6/ES2015	2015	let/const, classes, arrow functions, modules
ES2016	2016	includes(), opérateur **
ES2017	2017	async/await, Object.entries()
ES2018+	2018+	Rest/spread, optional chaining, etc.

Table 1: Principales versions d'ECMAScript

Important

Tous les navigateurs ne supportent pas les dernières fonctionnalités. Utilisez `https://caniuse.com` pour vérifier la compatibilité. Pour les projets en production, des outils comme **Babel** transpilent le code moderne en ES5.

1.3 Intégrer JavaScript dans un document HTML

1.3.1 Script interne

```html
<!DOCTYPE html>
<html lang="fr">
<head>
  <title>Ma page</title>
</head>
<body>
  <h1>Bonjour</h1>

  <!-- Script en fin de body (recommande) -->
  <script>
    console.log('Hello World!');
    alert('Bienvenue !');
  </script>
</body>
</html>
```

1.3.2 Script externe (recommandé)

```html
<!DOCTYPE html>
<html lang="fr">
<head>
  <title>Ma page</title>
  <!-- Avec defer : charge en parallele, execute apres le DOM -->
  <script src="script.js" defer></script>
</head>
<body>
  <h1>Bonjour</h1>
</body>
</html>
```

À retenir

Placement du script :

- **En fin de body** : méthode classique, le DOM est prêt
- **Dans head avec** `defer` : charge en parallèle, exécute après le DOM
- **Dans head avec** `async` : charge et exécute dès que possible (ordre non garanti)

1.4 La console JavaScript

La console est l'outil de développement essentiel pour tester et déboguer. Accès : **F12** puis onglet "Console".

Listing 1: Méthodes de la console

```
// Afficher un message
console.log('Message simple');
console.log('Nom:', nom, 'Age:', age);

// Niveaux de log
console.info('Information');
console.warn('Attention !');
console.error('Erreur !');

// Afficher un objet
console.log(monObjet);
console.dir(monObjet);        // Vue detaillee
console.table(monTableau);    // Affichage en tableau

// Mesurer le temps
console.time('timer');
// ... code a mesurer ...
console.timeEnd('timer');     // Affiche le temps ecoule

// Grouper des messages
console.group('Mon groupe');
console.log('Message 1');
console.log('Message 2');
console.groupEnd();

// Effacer la console
console.clear();
```

2 Les bases du JavaScript

2.1 Les variables : var, let et const

Définition

Une **variable** est un conteneur nommé qui stocke une valeur. En JavaScript moderne, on utilise `let` et `const`.

2.1.1 Déclaration avec let

```
// Declaration et assignation
let age = 25;
let nom = 'Alice';

// Declaration sans valeur initiale
let prenom;             // undefined
prenom = 'Bob';         // Assignation ulterieure

// Reassignation possible
age = 26;
nom = 'Charlie';
```

2.1.2 Déclaration avec const

```
// Constante : ne peut pas etre reassignee
const PI = 3.14159;
const URL_API = 'https://api.exemple.com';

// ERREUR : reassignation interdite
// PI = 3.14;  // TypeError!

// ATTENTION : pour les objets/tableaux, le contenu peut changer
const personne = { nom: 'Alice' };
personne.nom = 'Bob';      // OK : modification du contenu
// personne = {};          // ERREUR : reassignation

const nombres = [1, 2, 3];
nombres.push(4);           // OK : [1, 2, 3, 4]
// nombres = [];           // ERREUR
```

2.1.3 L'ancien var (à éviter)

```
// var a une portee de fonction (pas de bloc)
var x = 10;

if (true) {
  var y = 20;    // Accessible en dehors du bloc!
```

```
  let z = 30;    // Accessible uniquement dans le bloc
}

console.log(y); // 20 (fuite de portee!)
// console.log(z); // ReferenceError
```

Important

Règle moderne :

- Utilisez `const` par défaut
- Utilisez `let` si vous devez réassigner la variable
- N'utilisez jamais `var`

2.1.4 Règles de nommage

```
// Valide
let nom;
let _prive;
let $element;
let monNomComplet;          // camelCase (convention)
let nombre123;

// Invalide
// let 123nombre;           // Ne peut pas commencer par un chiffre
// let mon-nom;             // Tiret interdit
// let mon nom;             // Espace interdit
// let class;               // Mot reserve interdit
```

2.2 Les types en JavaScript

JavaScript possède des types **primitifs** et des types **objets**.

2.2.1 Types primitifs

```
// String (chaine de caracteres)
let nom = 'Alice';
let message = "Bonjour";
let template = `Bonjour ${nom}`;  // Template literal (ES6)

// Number (nombre entier ou decimal)
let age = 25;
let prix = 19.99;
let negatif = -10;
let infini = Infinity;
let pasUnNombre = NaN;     // Not a Number

// Boolean (booleen)
```

```
let estActif = true;
let estConnecte = false;

// Undefined (non defini)
let variable;              // undefined par defaut

// Null (absence intentionnelle de valeur)
let resultat = null;

// Symbol (identifiant unique - ES6)
let id = Symbol('id');

// BigInt (grands entiers - ES2020)
let grandNombre = 9007199254740991n;
```

2.2.2 L'opérateur typeof

```
typeof 'hello'       // 'string'
typeof 42            // 'number'
typeof true          // 'boolean'
typeof undefined     // 'undefined'
typeof null          // 'object' (bug historique!)
typeof {}            // 'object'
typeof []            // 'object'
typeof function(){} // 'function'
typeof Symbol()      // 'symbol'
```

2.2.3 Conversions de types

```
// Conversion en String
String(123);         // '123'
(123).toString();    // '123'
123 + '';            // '123' (concatenation)

// Conversion en Number
Number('42');        // 42
Number('3.14');      // 3.14
Number('hello');     // NaN
Number(true);        // 1
Number(false);       // 0
Number(null);        // 0
Number(undefined);   // NaN

parseInt('42px');    // 42 (ignore le reste)
parseFloat('3.14'); // 3.14

+'42';               // 42 (operateur unaire +)

// Conversion en Boolean
```

```
Boolean(1);           // true
Boolean(0);           // false
Boolean('hello');     // true
Boolean('');          // false
Boolean(null);        // false
Boolean(undefined);   // false

!!valeur;             // Double negation = conversion en boolean
```

À retenir

Valeurs "falsy" (évaluées à false) : `false`, `0`, `''`, `null`, `undefined`, `NaN`
Tout le reste est "truthy", y compris `[]`, `{}`, `'0'`, `'false'`

2.3 Les opérateurs de comparaison et logiques

2.3.1 Opérateurs de comparaison

```
// Egalite stricte (type + valeur) - RECOMMANDE
5 === 5          // true
5 === '5'        // false (types differents)
null === undefined  // false

// Egalite faible (avec conversion de type) - A EVITER
5 == '5'         // true (conversion automatique)
null == undefined   // true
0 == false       // true

// Inegalite
5 !== 3          // true (stricte)
5 != '5'         // false (faible)

// Comparaisons
5 > 3            // true
5 >= 5           // true
3 < 5            // true
3 <= 3           // true

// Comparaison de chaines (ordre alphabetique)
'a' < 'b'        // true
'abc' < 'abd' // true
```

Important

Utilisez **toujours** `===` et `!==` (égalité stricte). L'égalité faible (`==`) peut produire des résultats surprenants.

2.3.2 Opérateurs logiques

```
// ET logique (&&)
true && true     // true
true && false    // false
false && true    // false

// OU logique (||)
true || false    // true
false || true    // true
false || false   // false

// NON logique (!)
!true            // false
!false           // true
!0               // true
!'hello'         // false

// Court-circuit
// && retourne la premiere valeur falsy ou la derniere
let a = null && 'hello';    // null
let b = 'hi' && 'hello';    // 'hello'

// || retourne la premiere valeur truthy ou la derniere
let c = null || 'defaut';   // 'defaut'
let d = 'value' || 'def';   // 'value'

// Coalescence nulle (??) - ES2020
// Retourne la droite seulement si gauche est null/undefined
let e = null ?? 'defaut';   // 'defaut'
let f = 0 ?? 'defaut';      // 0 (0 n'est pas null/undefined)
let g = '' ?? 'defaut';     // '' (idem)
```

2.4 Valeur vs Référence

Définition

Les **primitifs** sont passés par **valeur** (copie). Les **objets** (et tableaux) sont passés par **référence** (même emplacement mémoire).

```
// PRIMITIFS : copie de la valeur
let a = 10;
let b = a;       // b est une copie
b = 20;
console.log(a); // 10 (inchange)
console.log(b); // 20

// OBJETS : copie de la reference
let obj1 = { nom: 'Alice' };
let obj2 = obj1;     // obj2 pointe vers le meme objet
obj2.nom = 'Bob';
```

```
12 console.log(obj1.nom); // 'Bob' (modifie aussi!)
13
14 // TABLEAUX : meme comportement
15 let arr1 = [1, 2, 3];
16 let arr2 = arr1;
17 arr2.push(4);
18 console.log(arr1); // [1, 2, 3, 4]
19
20 // Pour copier un objet/tableau (copie superficielle)
21 let copieObj = { ...obj1 };          // Spread operator
22 let copieArr = [...arr1];            // Spread operator
23 let copieArr2 = arr1.slice();        // Methode slice
24 let copieObj2 = Object.assign({}, obj1);
```

3 Les structures de contrôle

3.1 Les instructions conditionnelles

3.1.1 if / else if / else

```
let age = 18;

if (age < 13) {
  console.log('Enfant');
} else if (age < 18) {
  console.log('Adolescent');
} else if (age < 65) {
  console.log('Adulte');
} else {
  console.log('Senior');
}

// Forme courte (une seule instruction)
if (age >= 18) console.log('Majeur');

// Condition avec operateurs logiques
if (age >= 18 && age < 65) {
  console.log('Adulte actif');
}
```

3.1.2 L'opérateur ternaire

```
// Syntaxe : condition ? valeurSiVrai : valeurSiFaux
let statut = age >= 18 ? 'Majeur' : 'Mineur';

// Equivalent a :
let statut2;
if (age >= 18) {
  statut2 = 'Majeur';
} else {
  statut2 = 'Mineur';
}

// Ternaires imbriques (a eviter si trop complexe)
let categorie = age < 13 ? 'Enfant'
              : age < 18 ? 'Ado'
              : 'Adulte';
```

3.1.3 switch

```
let jour = 'lundi';

switch (jour) {
```

```
    case 'lundi':
    case 'mardi':
    case 'mercredi':
    case 'jeudi':
    case 'vendredi':
      console.log('Jour de semaine');
      break;
    case 'samedi':
    case 'dimanche':
      console.log('Week-end');
      break;
    default:
      console.log('Jour invalide');
}

// Attention : sans break, l'execution continue (fall-through)
```

3.2 Les boucles

3.2.1 Boucle for

```
// Boucle for classique
for (let i = 0; i < 5; i++) {
  console.log(i);  // 0, 1, 2, 3, 4
}

// Parcourir un tableau
const fruits = ['pomme', 'banane', 'orange'];
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}

// Boucle decroissante
for (let i = 10; i >= 0; i--) {
  console.log(i);
}

// Avec un pas different
for (let i = 0; i < 10; i += 2) {
  console.log(i);  // 0, 2, 4, 6, 8
}
```

3.2.2 Boucle for...of (ES6)

```
// Parcourir les VALEURS d'un iterable
const fruits = ['pomme', 'banane', 'orange'];

for (const fruit of fruits) {
  console.log(fruit);  // pomme, banane, orange
```

```
}

// Fonctionne aussi sur les chaines
for (const lettre of 'Bonjour') {
  console.log(lettre);  // B, o, n, j, o, u, r
}
```

3.2.3 Boucle for...in

```
// Parcourir les CLES/INDEX d'un objet
const personne = { nom: 'Alice', age: 25 };

for (const cle in personne) {
  console.log(cle, personne[cle]);
  // nom Alice
  // age 25
}

// Sur un tableau : parcourt les INDEX (a eviter)
const arr = ['a', 'b', 'c'];
for (const index in arr) {
  console.log(index);  // '0', '1', '2' (chaines!)
}
```

Important

Utilisez `for...of` pour les tableaux et `for...in` pour les objets. Ne mélangez pas !

3.2.4 Boucles while et do...while

```
// while : condition verifiee AVANT
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}

// do...while : condition verifiee APRES (au moins 1 execution)
let j = 0;
do {
  console.log(j);
  j++;
} while (j < 5);

// Exemple pratique : attendre une condition
let tentatives = 0;
while (tentatives < 3) {
  // ... essayer quelque chose
```

```
    tentatives++;
}
```

3.2.5 break et continue

```
// break : sort completement de la boucle
for (let i = 0; i < 10; i++) {
  if (i === 5) break;
  console.log(i);  // 0, 1, 2, 3, 4
}

// continue : passe a l'iteration suivante
for (let i = 0; i < 5; i++) {
  if (i === 2) continue;
  console.log(i);  // 0, 1, 3, 4 (2 est saute)
}
```

4 Les nombres

4.1 Représentation des nombres

```
// Entiers et decimaux
let entier = 42;
let decimal = 3.14;
let negatif = -10;

// Notation scientifique
let grand = 1e6;      // 1000000
let petit = 1e-6;     // 0.000001

// Autres bases
let binaire = 0b1010;     // 10 en decimal
let octal = 0o755;        // 493 en decimal
let hexa = 0xFF;          // 255 en decimal

// Valeurs speciales
let infini = Infinity;
let moinsInfini = -Infinity;
let pasNombre = NaN;      // Not a Number

// Verification
isNaN(NaN);               // true
isNaN('hello');           // true (converti d'abord)
Number.isNaN(NaN);        // true (plus strict)
Number.isNaN('hello');    // false

isFinite(100);            // true
isFinite(Infinity);       // false
```

Important

Les nombres décimaux en JavaScript peuvent avoir des problèmes de précision :

```
0.1 + 0.2;                // 0.30000000000000004 !
0.1 + 0.2 === 0.3;        // false !

// Solution : arrondir ou utiliser des entiers
(0.1 + 0.2).toFixed(2);   // '0.30'
Math.round((0.1 + 0.2) * 100) / 100;   // 0.3
```

4.2 Méthodes de Number

```
let n = 123.456;

// Conversion en chaine
n.toString();         // '123.456'
```

```
n.toString(2);        // '1111011' (binaire)
n.toString(16);       // '7b' (hexadecimal)

// Formatage
n.toFixed(2);         // '123.46' (2 decimales, arrondi)
n.toFixed(0);         // '123'
n.toPrecision(4);     // '123.5' (4 chiffres significatifs)
n.toExponential(2);   // '1.23e+2'

// Methodes statiques
Number.isInteger(42);       // true
Number.isInteger(42.0);     // true
Number.isInteger(42.5);     // false

Number.parseInt('42px');    // 42
Number.parseFloat('3.14'); // 3.14

Number.MAX_VALUE;           // Plus grand nombre representable
Number.MIN_VALUE;           // Plus petit nombre positif
Number.MAX_SAFE_INTEGER;    // 9007199254740991
Number.MIN_SAFE_INTEGER;    // -9007199254740991
```

4.3 L'objet Math

```
// Constantes
Math.PI;            // 3.1415926535589793
Math.E;             // 2.718281828459045

// Arrondis
Math.round(4.5);    // 5 (au plus proche)
Math.floor(4.9);    // 4 (vers le bas)
Math.ceil(4.1);     // 5 (vers le haut)
Math.trunc(4.9);    // 4 (supprime les decimales)

// Valeur absolue
Math.abs(-5);       // 5

// Puissance et racine
Math.pow(2, 3);     // 8 (2^3)
2 ** 3;             // 8 (operateur ES2016)
Math.sqrt(16);      // 4
Math.cbrt(27);      // 3 (racine cubique)

// Min / Max
Math.min(1, 5, 3);        // 1
Math.max(1, 5, 3);        // 5
Math.min(...tableau);     // Min d'un tableau

// Aleatoire
Math.random();            // Entre 0 (inclus) et 1 (exclus)
```

```
27
28 // Nombre aleatoire entre min et max (inclus)
29 function randomInt(min, max) {
30   return Math.floor(Math.random() * (max - min + 1)) + min;
31 }
32 randomInt(1, 10);        // Entre 1 et 10
33
34 // Trigonometrie
35 Math.sin(Math.PI / 2);   // 1
36 Math.cos(0);             // 1
37 Math.tan(0);             // 0
38
39 // Logarithmes
40 Math.log(Math.E);        // 1 (log naturel)
41 Math.log10(100);         // 2
42 Math.log2(8);            // 3
```

5 Les chaînes de caractères

5.1 Création et syntaxe

```
// Guillemets simples ou doubles
let str1 = 'Hello';
let str2 = "World";

// Template literals (ES6) - backticks
let nom = 'Alice';
let message = `Bonjour ${nom}!`;  // Interpolation

// Chaines multi-lignes
let multiLigne = `Ligne 1
Ligne 2
Ligne 3`;

// Caracteres d'echappement
let special = 'C\'est une apostrophe';
let tabulation = 'Col1\tCol2';
let nouvelleLigne = 'Ligne1\nLigne2';
let backslash = 'Chemin\\fichier';
```

5.2 Propriétés et accès aux caractères

```
let str = 'Bonjour';

// Longueur
str.length;            // 7

// Acces par index (lecture seule)
str[0];                // 'B'
str[6];                // 'r'
str[10];               // undefined
str.charAt(0);         // 'B'
str.at(-1);            // 'r' (ES2022, index negatif)

// Code caractere
str.charCodeAt(0);     // 66 (code ASCII de 'B')
String.fromCharCode(66);  // 'B'
```

Important

Les chaînes sont **immuables** en JavaScript. Les méthodes retournent de nouvelles chaînes sans modifier l'originale.

5.3 Méthodes sur les chaînes

```
let str = '  Hello World  ';

// Modification de casse
str.toUpperCase();          // '  HELLO WORLD  '
str.toLowerCase();          // '  hello world  '

// Suppression des espaces
str.trim();                 // 'Hello World'
str.trimStart();            // 'Hello World  '
str.trimEnd();              // '  Hello World'

// Recherche
let texte = 'JavaScript est genial';
texte.indexOf('Script');      // 4 (position)
texte.indexOf('Python');      // -1 (non trouve)
texte.lastIndexOf('a');       // 19 (derniere occurrence)
texte.includes('Script');     // true
texte.startsWith('Java');     // true
texte.endsWith('genial');     // true

// Extraction
texte.slice(0, 4);            // 'Java'
texte.slice(4);               // 'Script est genial'
texte.slice(-6);              // 'genial' (depuis la fin)
texte.substring(0, 4);        // 'Java'

// Remplacement
texte.replace('genial', 'super');    // Premier seulement
texte.replaceAll('a', '@');          // Tous les 'a'

// Division et jointure
'a,b,c'.split(',');           // ['a', 'b', 'c']
'hello'.split('');            // ['h', 'e', 'l', 'l', 'o']
['a', 'b', 'c'].join('-');    // 'a-b-c'

// Repetition et padding
'ab'.repeat(3);               // 'ababab'
'5'.padStart(3, '0');         // '005'
'5'.padEnd(3, '0');           // '500'
```

5.4 Les expressions régulières

```
// Creation d'une regex
let regex1 = /motif/;              // Literal
let regex2 = new RegExp('motif');

// Flags
/motif/i;     // i = insensible a la casse
/motif/g;     // g = global (toutes les occurrences)
/motif/m;     // m = multiligne
```

```
9
10 // Methodes de test
11 let texte = 'Bonjour tout le monde';
12 /bonjour/i.test(texte);       // true
13 texte.match(/o/g);            // ['o', 'o', 'o', 'o']
14 texte.search(/tout/);         // 8 (position)
15 texte.replace(/o/g, '0');     // 'B0nj0ur t0ut le m0nde'
16
17 // Metacaracteres courants
18 // .      n'importe quel caractere
19 // \d     chiffre [0-9]
20 // \w     mot [a-zA-Z0-9_]
21 // \s     espace blanc
22 // ^      debut de chaine
23 // $      fin de chaine
24 // +      1 ou plus
25 // *      0 ou plus
26 // ?      0 ou 1
27 // {n,m} entre n et m fois
28
29 // Exemples pratiques
30 /^\d{5}$/.test('75001');          // Code postal
31 /^[\w.-]+@[\w.-]+\.\w+$/.test('a@b.com');  // Email simple
32 /^0[67]\d{8}$/.test('0612345678');  // Tel mobile FR
```

6 Les objets

6.1 Introduction aux objets

Définition

Un **objet** est une collection de paires clé-valeur. C'est la structure de données la plus utilisée en JavaScript pour représenter des entités complexes.

```
// Creation d'un objet litteral
const personne = {
  prenom: 'Alice',
  nom: 'Dupont',
  age: 25,
  estEtudiant: true,
  adresse: {
    ville: 'Paris',
    codePostal: '75001'
  }
};

// Acces aux proprietes
personne.prenom;            // 'Alice' (notation point)
personne['nom'];            // 'Dupont' (notation crochet)

// Notation crochet necessaire pour :
let prop = 'age';
personne[prop];             // 25 (cle dynamique)
personne['code-postal'];    // Cle avec caracteres speciaux

// Modification
personne.age = 26;
personne['ville'] = 'Lyon';

// Ajout de propriete
personne.email = 'alice@example.com';

// Raccourcis ES6
let x = 10, y = 20;
let point = { x, y };       // Equivalent a { x: x, y: y }

// Methodes (fonctions dans un objet)
const calculatrice = {
  a: 0,
  b: 0,
  additionner() {
    return this.a + this.b;
  },
  multiplier: function() {
    return this.a * this.b;
  }
```

```
};
```

6.2 Manipuler les propriétés

```
const obj = { a: 1, b: 2, c: 3 };

// Verifier l'existence d'une propriete
'a' in obj;                    // true
obj.hasOwnProperty('a');       // true

// Supprimer une propriete
delete obj.c;
console.log(obj);              // { a: 1, b: 2 }

// Lister les proprietes
Object.keys(obj);              // ['a', 'b']
Object.values(obj);            // [1, 2]
Object.entries(obj);           // [['a', 1], ['b', 2]]

// Nombre de proprietes
Object.keys(obj).length;       // 2
```

6.3 Fusionner, comparer et itérer

```
// Fusion d'objets
const defauts = { theme: 'clair', langue: 'fr' };
const options = { theme: 'sombre' };

// Spread operator (ES6)
const config = { ...defauts, ...options };
// { theme: 'sombre', langue: 'fr' }

// Object.assign
const config2 = Object.assign({}, defauts, options);

// Comparaison d'objets
const obj1 = { a: 1 };
const obj2 = { a: 1 };
obj1 === obj2;                 // false (references differentes)

// Comparaison par contenu
JSON.stringify(obj1) === JSON.stringify(obj2);  // true

// Iteration sur un objet
const personne = { nom: 'Alice', age: 25 };

for (const cle in personne) {
  console.log(`${cle}: ${personne[cle]}`);
}
```

```
26
27 // Avec Object.entries
28 for (const [cle, valeur] of Object.entries(personne)) {
29   console.log(`${cle}: ${valeur}`);
30 }
```

6.4 Le format JSON

Définition

JSON (JavaScript Object Notation) est un format textuel pour représenter des données structurées. Il est utilisé pour échanger des données entre un serveur et une application web.

```
1  // Objet JavaScript vers JSON (serialisation)
2  const personne = {
3    nom: 'Alice',
4    age: 25,
5    hobbies: ['lecture', 'sport']
6  };
7
8  const json = JSON.stringify(personne);
9  // '{"nom":"Alice","age":25,"hobbies":["lecture","sport"]}'
10
11 // Avec indentation pour lisibilite
12 JSON.stringify(personne, null, 2);
13
14 // JSON vers objet JavaScript (deserialisation)
15 const jsonStr = '{"nom":"Bob","age":30}';
16 const obj = JSON.parse(jsonStr);
17 console.log(obj.nom);  // 'Bob'
18
19 // Attention aux erreurs de parsing
20 try {
21   JSON.parse('invalid json');
22 } catch (e) {
23   console.error('JSON invalide');
24 }
```

Important

JSON ne supporte pas : les fonctions, `undefined`, les dates (converties en chaînes), les références circulaires.

7 Les fonctions

7.1 Déclaration et expression de fonction

```
// Declaration de fonction (hoisted)
function saluer(nom) {
  return `Bonjour ${nom}!`;
}

// Expression de fonction (non hoisted)
const saluer2 = function(nom) {
  return `Bonjour ${nom}!`;
};

// Fonction anonyme (utilisee comme callback)
setTimeout(function() {
  console.log('Apres 1 seconde');
}, 1000);

// Appel de fonction
saluer('Alice');          // 'Bonjour Alice!'
saluer2('Bob');           // 'Bonjour Bob!'
```

7.2 Paramètres et arguments

```
// Parametres simples
function additionner(a, b) {
  return a + b;
}
additionner(2, 3);        // 5
additionner(2);           // NaN (b = undefined)

// Parametres par defaut (ES6)
function saluer(nom = 'inconnu', ponctuation = '!') {
  return `Bonjour ${nom}${ponctuation}`;
}
saluer();                 // 'Bonjour inconnu!'
saluer('Alice');          // 'Bonjour Alice!'
saluer('Bob', '?');       // 'Bonjour Bob?'

// Objet arguments (ancien, a eviter)
function ancienneMethode() {
  console.log(arguments);        // Objet pseudo-tableau
  console.log(arguments[0]);
  console.log(arguments.length);
}

// Operateur rest (ES6, prefere)
function nouvelleMethode(...args) {
  console.log(args);             // Vrai tableau
```

```
  return args.reduce((a, b) => a + b, 0);
}
nouvelleMethode(1, 2, 3, 4);    // 10

// Combinaison parametres normaux et rest
function afficher(premier, ...autres) {
  console.log('Premier:', premier);
  console.log('Autres:', autres);
}
afficher('a', 'b', 'c');
// Premier: a
// Autres: ['b', 'c']
```

7.3 Valeur de retour

```
// Retour simple
function carre(n) {
  return n * n;
}

// Retour implicite (undefined)
function sansRetour() {
  console.log('Pas de return');
}
let resultat = sansRetour();  // undefined

// Retour anticipe
function verifier(age) {
  if (age < 0) return 'Age invalide';
  if (age < 18) return 'Mineur';
  return 'Majeur';
}

// Retour d'objet
function creerPersonne(nom, age) {
  return { nom, age };
}

// Retour de tableau
function minMax(arr) {
  return [Math.min(...arr), Math.max(...arr)];
}
const [min, max] = minMax([3, 1, 4, 1, 5]);
```

7.4 Les fonctions fléchées (arrow functions)

```
// Syntaxe complete
const additionner = (a, b) => {
  return a + b;
```

```
};

// Retour implicite (sans accolades)
const additionner2 = (a, b) => a + b;

// Un seul parametre : parentheses optionnelles
const carre = n => n * n;

// Pas de parametre : parentheses obligatoires
const direBonjour = () => 'Bonjour!';

// Retour d'objet (parentheses necessaires)
const creerPoint = (x, y) => ({ x, y });

// Utilisation courante avec les methodes de tableau
const nombres = [1, 2, 3, 4, 5];
const doubles = nombres.map(n => n * 2);
const pairs = nombres.filter(n => n % 2 === 0);
const somme = nombres.reduce((acc, n) => acc + n, 0);
```

Important

Les fonctions fléchées n'ont pas leur propre `this`. Elles héritent du `this` du contexte englobant. Évitez-les pour les méthodes d'objet qui utilisent `this`.

7.5 Callbacks et closures

7.5.1 Fonctions de rappel (callbacks)

```
// Une callback est une fonction passee en argument
function executerApres(callback) {
  console.log('Avant');
  callback();
  console.log('Apres');
}

executerApres(() => console.log('Pendant'));
// Avant
// Pendant
// Apres

// Exemples courants
setTimeout(() => console.log('Apres 1s'), 1000);

document.addEventListener('click', (event) => {
  console.log('Click!', event);
});

const nombres = [1, 2, 3];
nombres.forEach((n, index) => {
```

```
  console.log(`Index ${index}: ${n}`);
});
```

7.5.2 Fermetures (closures)

```
// Une closure est une fonction qui "capture" son environnement
function creerCompteur() {
  let compte = 0;  // Variable privee

  return function() {
    compte++;
    return compte;
  };
}

const compteur1 = creerCompteur();
compteur1();  // 1
compteur1();  // 2
compteur1();  // 3

const compteur2 = creerCompteur();
compteur2();  // 1 (independant de compteur1)

// Exemple pratique : fonction de multiplication
function multiplierPar(facteur) {
  return (nombre) => nombre * facteur;
}

const double = multiplierPar(2);
const triple = multiplierPar(3);

double(5);    // 10
triple(5);    // 15
```

8 Les tableaux

8.1 Création et accès

```
// Creation
const vide = [];
const nombres = [1, 2, 3, 4, 5];
const mixte = [1, 'deux', true, null, { a: 1 }];

// Acces par index (commence a 0)
nombres[0];        // 1
nombres[4];        // 5
nombres[10];       // undefined
nombres.at(-1);    // 5 (ES2022, dernier element)

// Modification
nombres[0] = 10;
nombres[10] = 100;  // Cree des "trous"

// Proprietes
nombres.length;    // 11 (apres l'ajout ci-dessus)

// Verification
Array.isArray(nombres);  // true
Array.isArray('abc');    // false
```

8.2 Ajouter et supprimer des éléments

```
const arr = [1, 2, 3];

// A la fin
arr.push(4);            // [1, 2, 3, 4] - retourne nouvelle
   longueur
arr.push(5, 6);         // [1, 2, 3, 4, 5, 6]

// Au debut
arr.unshift(0);         // [0, 1, 2, 3, 4, 5, 6]

// Supprimer a la fin
arr.pop();              // Retourne 6, arr = [0, 1, 2, 3, 4, 5]

// Supprimer au debut
arr.shift();            // Retourne 0, arr = [1, 2, 3, 4, 5]

// Supprimer/inserer avec splice
arr.splice(2, 1);       // Supprime 1 element a l'index 2
                        // arr = [1, 2, 4, 5]

arr.splice(2, 0, 3);    // Insere 3 a l'index 2, supprime 0
                        // arr = [1, 2, 3, 4, 5]
```

```
22
23 arr.splice(1, 2, 'a', 'b');  // Remplace 2 elements par 'a' et 'b'
24                              // arr = [1, 'a', 'b', 4, 5]
```

8.3 Rechercher dans un tableau

```
1  const fruits = ['pomme', 'banane', 'orange', 'banane'];
2
3  // Trouver l'index
4  fruits.indexOf('banane');       // 2 (premier trouve)
5  fruits.lastIndexOf('banane');   // 3 (dernier trouve)
6  fruits.indexOf('kiwi');         // -1 (non trouve)
7
8  // Verifier la presence
9  fruits.includes('orange');      // true
10 fruits.includes('kiwi');        // false
11
12 // Trouver avec condition
13 const nombres = [1, 5, 10, 15, 20];
14
15 nombres.find(n => n > 8);       // 10 (premier qui satisfait)
16 nombres.findIndex(n => n > 8); // 2 (index du premier)
17 nombres.findLast(n => n > 8);  // 20 (dernier - ES2023)
18
19 // Verifier si au moins un / tous satisfont
20 nombres.some(n => n > 15);     // true (au moins un)
21 nombres.every(n => n > 0);     // true (tous)
```

8.4 Transformer et itérer

```
1  const nombres = [1, 2, 3, 4, 5];
2
3  // forEach : executer pour chaque element
4  nombres.forEach((n, index) => {
5    console.log(`Index ${index}: ${n}`);
6  });
7
8  // map : transformer chaque element
9  const doubles = nombres.map(n => n * 2);
10 // [2, 4, 6, 8, 10]
11
12 // filter : garder les elements qui satisfont une condition
13 const pairs = nombres.filter(n => n % 2 === 0);
14 // [2, 4]
15
16 // reduce : reduire a une seule valeur
17 const somme = nombres.reduce((acc, n) => acc + n, 0);
18 // 15
```

```
19
20 const max = nombres.reduce((acc, n) => Math.max(acc, n), -
       Infinity);
21 // 5
22
23 // Enchainement de methodes
24 const resultat = nombres
25   .filter(n => n % 2 === 1)     // Garder impairs
26   .map(n => n * 2)               // Doubler
27   .reduce((a, b) => a + b, 0);  // Sommer
28 // (1*2) + (3*2) + (5*2) = 18
```

8.5 Trier et fusionner

```
1  // Tri (modifie le tableau original!)
2  const arr = [3, 1, 4, 1, 5];
3  arr.sort();                    // [1, 1, 3, 4, 5] (alphabetique!)
4
5  // Attention : tri alphabetique par defaut
6  [10, 2, 1].sort();             // [1, 10, 2] (!!)
7
8  // Tri numerique
9  arr.sort((a, b) => a - b);     // Croissant
10 arr.sort((a, b) => b - a);     // Decroissant
11
12 // Tri d'objets
13 const personnes = [
14   { nom: 'Alice', age: 30 },
15   { nom: 'Bob', age: 25 }
16 ];
17 personnes.sort((a, b) => a.age - b.age);
18
19 // Inverser
20 arr.reverse();
21
22 // Fusionner
23 const a = [1, 2];
24 const b = [3, 4];
25 const c = a.concat(b);         // [1, 2, 3, 4]
26 const d = [...a, ...b];        // [1, 2, 3, 4] (spread)
27
28 // Aplatir
29 const nested = [1, [2, 3], [4, [5, 6]]];
30 nested.flat();                 // [1, 2, 3, 4, [5, 6]]
31 nested.flat(2);                // [1, 2, 3, 4, 5, 6]
32
33 // flatMap
34 [1, 2, 3].flatMap(n => [n, n * 2]);
35 // [1, 2, 2, 4, 3, 6]
```

8.6 Décomposition (destructuring)

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

// Decomposition simple
const [premier, deuxieme] = nombres;
console.log(premier);     // 1
console.log(deuxieme);    // 2

// Ignorer des elements
const [a, , c] = nombres;  // a=1, c=3

// Avec rest
const [tete, ...reste] = nombres;
console.log(tete);        // 1
console.log(reste);       // [2, 3, 4, 5]

// Valeurs par defaut
const [x, y, z = 0] = [1, 2];
console.log(z);           // 0

// Echange de variables
let m = 1, n = 2;
[m, n] = [n, m];          // m=2, n=1
```

9 Les modules

9.1 Introduction aux modules ES6

Définition

Les **modules** permettent de diviser le code en fichiers séparés, chacun avec son propre scope. Cela améliore l'organisation, la réutilisation et la maintenance du code.

Pour utiliser les modules dans le navigateur :

```
<script type="module" src="main.js"></script>
```

9.2 Export et Import

9.2.1 Exports nommés

```
// fichier: math.js

// Export individuel
export const PI = 3.14159;

export function additionner(a, b) {
  return a + b;
}

export function multiplier(a, b) {
  return a * b;
}

// Ou export groupe a la fin
const E = 2.71828;
function soustraire(a, b) { return a - b; }
export { E, soustraire };
```

```
// fichier: main.js

// Import nomme (accolades obligatoires)
import { PI, additionner } from './math.js';

console.log(PI);                // 3.14159
console.log(additionner(2, 3)); // 5

// Renommer a l'import
import { multiplier as mult } from './math.js';
mult(2, 3);  // 6

// Importer tout
import * as MathUtils from './math.js';
MathUtils.PI;
```

```
MathUtils.additionner(2, 3);
```

9.2.2 Export par défaut

```
// fichier: Personne.js

// Un seul export default par fichier
export default class Personne {
  constructor(nom) {
    this.nom = nom;
  }

  saluer() {
    return `Bonjour, je suis ${this.nom}`;
  }
}

// Peut coexister avec des exports nommes
export const VERSION = '1.0';
```

```
// fichier: main.js

// Import default (pas d'accolades, nom au choix)
import Personne from './Personne.js';
import MaClasse from './Personne.js';  // Meme chose

const alice = new Personne('Alice');

// Combiner default et nommes
import Personne, { VERSION } from './Personne.js';
```

9.3 Réexport et imports dynamiques

```
// fichier: index.js (barrel file)
// Reexporte tout depuis un point central

export { additionner, multiplier } from './math.js';
export { default as Personne } from './Personne.js';
export * from './utils.js';  // Tout sauf default
```

```
// Import dynamique (chargement a la demande)
async function chargerModule() {
  const module = await import('./math.js');
  console.log(module.additionner(2, 3));
}

// Ou avec then
import('./math.js').then(module => {
  console.log(module.PI);
```

```
});

// Utile pour le code splitting
button.addEventListener('click', async () => {
  const { fonctionLourde } = await import('./heavy.js');
  fonctionLourde();
});
```

10 Le DOM

10.1 Introduction au DOM

Définition

Le **DOM** (Document Object Model) est une représentation arborescente du document HTML. JavaScript peut lire et modifier cette structure pour créer des pages dynamiques.

- **DOM** : représentation du document HTML
- **BOM** (Browser Object Model) : objets du navigateur (window, navigator, location...)
- **window** : objet global contenant tout (DOM, BOM, variables globales...)

```
// L'objet window est global
window.document;          // Le DOM
window.location;          // URL actuelle
window.navigator;         // Infos navigateur
window.innerWidth;        // Largeur viewport

// Les proprietes de window sont accessibles directement
document === window.document;  // true
alert('Hello');                // window.alert()
```

10.2 Sélectionner des éléments

```
// Par ID (retourne un seul element ou null)
const header = document.getElementById('header');

// Par classe (retourne HTMLCollection, live)
const boutons = document.getElementsByClassName('btn');

// Par nom de balise
const paragraphes = document.getElementsByTagName('p');

// Par selecteur CSS (RECOMMANDE)
const premier = document.querySelector('.btn');       // Premier
const tous = document.querySelectorAll('.btn');       // Tous (
    NodeList)

// Exemples de selecteurs
document.querySelector('#menu');            // ID
document.querySelector('.card');            // Classe
document.querySelector('div.container');    // Balise + classe
document.querySelector('[data-id="123"]'); // Attribut
document.querySelector('ul > li:first-child'); // Selecteur
    complexe

```

```
// Parcourir les resultats
document.querySelectorAll('.item').forEach(item => {
  console.log(item.textContent);
});
```

10.3 Modifier le contenu

```
const element = document.querySelector('#mon-element');

// Contenu textuel (securise, pas de HTML)
element.textContent = 'Nouveau texte';
console.log(element.textContent);

// Contenu HTML (attention aux injections XSS!)
element.innerHTML = '<strong>Texte en gras</strong>';

// HTML externe (inclut l'element lui-meme)
console.log(element.outerHTML);

// Inserer du HTML adjacent
element.insertAdjacentHTML('beforebegin', '<p>Avant</p>');
element.insertAdjacentHTML('afterbegin', '<p>Debut</p>');
element.insertAdjacentHTML('beforeend', '<p>Fin</p>');
element.insertAdjacentHTML('afterend', '<p>Apres</p>');
```

10.4 Attributs et propriétés

```
const input = document.querySelector('input');
const lien = document.querySelector('a');

// Attributs HTML (getAttribute/setAttribute)
lien.getAttribute('href');
lien.setAttribute('href', 'https://example.com');
lien.removeAttribute('target');
lien.hasAttribute('href');          // true

// Proprietes DOM (acces direct)
lien.href;                          // URL complete
lien.href = 'https://google.com';

// data-* attributes
// <div data-user-id="123" data-role="admin">
const div = document.querySelector('[data-user-id]');
div.dataset.userId;                 // '123'
div.dataset.role;                   // 'admin'
div.dataset.newProp = 'value';      // Ajoute data-new-prop
```

10.5 Modifier les styles et classes

```
const element = document.querySelector('.box');

// Style inline (proprietes en camelCase)
element.style.backgroundColor = 'red';
element.style.fontSize = '20px';
element.style.display = 'none';

// Lire le style calcule
const styles = getComputedStyle(element);
styles.backgroundColor;

// Manipulation des classes (RECOMMANDE)
element.classList.add('active');
element.classList.remove('hidden');
element.classList.toggle('visible');
element.classList.contains('active');  // true/false
element.classList.replace('old', 'new');

// Plusieurs classes
element.classList.add('class1', 'class2');
element.classList.remove('class1', 'class2');

// Ancienne methode (a eviter)
element.className = 'class1 class2';
```

10.6 Créer et manipuler des nœuds

```
// Creer un element
const div = document.createElement('div');
div.id = 'ma-div';
div.className = 'container';
div.textContent = 'Contenu';

// Creer un noeud texte
const texte = document.createTextNode('Mon texte');

// Ajouter au DOM
const parent = document.querySelector('#parent');
parent.appendChild(div);            // A la fin
parent.prepend(div);                // Au debut
parent.append(div, texte);          // Plusieurs elements

// Inserer par rapport a un element
const reference = document.querySelector('#ref');
parent.insertBefore(div, reference);

// Methodes modernes
reference.before(div);              // Avant
```

```
reference.after(div);                    // Apres
reference.replaceWith(div);              // Remplacer

// Supprimer
div.remove();                            // Se supprime lui-meme
parent.removeChild(div);                 // Supprime un enfant

// Cloner
const clone = div.cloneNode(true); // true = avec enfants
```

Exemple

Créer une liste dynamiquement :

```
const fruits = ['Pomme', 'Banane', 'Orange'];
const ul = document.createElement('ul');

fruits.forEach(fruit => {
  const li = document.createElement('li');
  li.textContent = fruit;
  ul.appendChild(li);
});

document.body.appendChild(ul);
```

11 Les événements

11.1 Introduction aux événements

Les événements permettent de réagir aux actions de l'utilisateur : clics, saisie clavier, survol, etc.

11.1.1 Méthode HTML (à éviter)

```
<button onclick="alert('Click!')">Cliquer</button>
<input onchange="handleChange(this.value)">
```

11.1.2 Propriétés on* du DOM

```
const btn = document.querySelector('button');

btn.onclick = function(event) {
  console.log('Click!');
};

// Inconvenient : un seul gestionnaire par evenement
btn.onclick = function() { console.log('Premier'); };
btn.onclick = function() { console.log('Deuxieme'); };
// Seul "Deuxieme" s'affichera
```

11.2 addEventListener (recommandé)

```
const btn = document.querySelector('button');

// Syntaxe de base
btn.addEventListener('click', function(event) {
  console.log('Click!');
});

// Avec fonction flechee
btn.addEventListener('click', (e) => {
  console.log('Position:', e.clientX, e.clientY);
});

// Fonction nommee (necessaire pour removeEventListener)
function handleClick(e) {
  console.log('Click!');
}
btn.addEventListener('click', handleClick);

// Plusieurs gestionnaires possibles
btn.addEventListener('click', handler1);
btn.addEventListener('click', handler2);  // Les deux s'executent

```

```
// Options
btn.addEventListener('click', handler, {
  once: true,       // S'execute une seule fois
  capture: true,    // Phase de capture
  passive: true     // N'appellera pas preventDefault()
});
```

11.3 Supprimer et déclencher des événements

```
// Supprimer un gestionnaire (fonction nommee obligatoire)
btn.removeEventListener('click', handleClick);

// Declencher un evenement manuellement
btn.click();  // Simule un clic

// Creer et dispatcher un evenement custom
const event = new CustomEvent('monEvent', {
  detail: { message: 'Hello!' }
});
btn.dispatchEvent(event);

btn.addEventListener('monEvent', (e) => {
  console.log(e.detail.message);
});
```

11.4 Événements courants

```
// Souris
element.addEventListener('click', handler);
element.addEventListener('dblclick', handler);
element.addEventListener('mouseenter', handler);  // Survol entre
element.addEventListener('mouseleave', handler);  // Survol sort
element.addEventListener('mousemove', handler);   // Mouvement

// Clavier
input.addEventListener('keydown', handler);   // Touche enfoncee
input.addEventListener('keyup', handler);     // Touche relachee
input.addEventListener('keypress', handler);  // Deprecated

// Formulaires
input.addEventListener('input', handler);     // Saisie en cours
input.addEventListener('change', handler);    // Valeur modifiee
    (blur)
input.addEventListener('focus', handler);     // Focus obtenu
input.addEventListener('blur', handler);      // Focus perdu
form.addEventListener('submit', handler);     // Soumission

// Document
```

```
document.addEventListener('DOMContentLoaded', handler); // DOM pret
window.addEventListener('load', handler);      // Page entiere chargee
window.addEventListener('resize', handler);    // Redimensionnement
window.addEventListener('scroll', handler);    // Defilement
```

11.5 L'objet event

```
element.addEventListener('click', function(event) {
  // Informations generales
  event.type;            // 'click'
  event.target;          // Element qui a declenche
  event.currentTarget;   // Element qui ecoute (this)
  event.timeStamp;       // Quand

  // Position souris
  event.clientX;         // X par rapport au viewport
  event.clientY;         // Y par rapport au viewport
  event.pageX;           // X par rapport a la page
  event.pageY;           // Y par rapport a la page

  // Touches modificatrices
  event.ctrlKey;         // Ctrl enfonce?
  event.shiftKey;        // Shift enfonce?
  event.altKey;          // Alt enfonce?

  // Pour les evenements clavier
  event.key;             // 'Enter', 'a', 'Escape'...
  event.code;            // 'KeyA', 'Enter'...
});
```

11.6 Bouillonnement et capture

```
// Les evenements se propagent :
// 1. Capture : window -> document -> ... -> target
// 2. Target : l'element clique
// 3. Bouillonnement : target -> ... -> document -> window

// Par defaut, les gestionnaires s'executent au bouillonnement
parent.addEventListener('click', () => console.log('parent'));
enfant.addEventListener('click', () => console.log('enfant'));
// Clic sur enfant : "enfant" puis "parent"

// Ecouter en phase de capture
parent.addEventListener('click', handler, true);
// ou
parent.addEventListener('click', handler, { capture: true });
```

```
15
16 // Stopper la propagation
17 enfant.addEventListener('click', (e) => {
18   e.stopPropagation();  // N'atteint pas le parent
19 });
```

11.7 Empêcher le comportement par défaut

```
1  // Empecher la soumission d'un formulaire
2  form.addEventListener('submit', (e) => {
3    e.preventDefault();
4    // Validation personnalisee...
5  });
6
7  // Empecher un lien de naviguer
8  lien.addEventListener('click', (e) => {
9    e.preventDefault();
10   console.log('Navigation bloquee');
11 });
12
13 // Empecher le menu contextuel
14 element.addEventListener('contextmenu', (e) => {
15   e.preventDefault();
16   // Menu personnalise...
17 });
```

Exemple

Délégation d'événements : Utiliser le bouillonnement pour gérer les événements d'éléments dynamiques.

```
1 // Au lieu d'ajouter un listener sur chaque bouton...
2 document.querySelector('#liste').addEventListener('click', (
    e) => {
3   if (e.target.matches('.btn-supprimer')) {
4     const item = e.target.closest('li');
5     item.remove();
6   }
7 });
```

12 Date

12.1 Créer une date

```
// Date actuelle
const maintenant = new Date();

// A partir d'une chaine
const date1 = new Date('2025-06-15');
const date2 = new Date('2025-06-15T14:30:00');
const date3 = new Date('June 15, 2025 14:30:00');

// A partir de composants (mois de 0 a 11!)
const date4 = new Date(2025, 5, 15);        // 15 juin 2025
const date5 = new Date(2025, 5, 15, 14, 30, 0);

// A partir d'un timestamp (millisecondes depuis 1970)
const date6 = new Date(1718456400000);

// Obtenir le timestamp actuel
Date.now();
```

Important

Les mois sont indexés de 0 à 11 ! Janvier = 0, Décembre = 11.

12.2 Manipuler les dates

```
const date = new Date('2025-06-15T14:30:00');

// Getters
date.getFullYear();     // 2025
date.getMonth();        // 5 (juin, car 0-indexed)
date.getDate();         // 15 (jour du mois)
date.getDay();          // 0 (dimanche) a 6 (samedi)
date.getHours();        // 14
date.getMinutes();      // 30
date.getSeconds();      // 0
date.getMilliseconds();// 0
date.getTime();         // Timestamp en ms

// Setters
date.setFullYear(2026);
date.setMonth(11);      // Decembre
date.setDate(25);
date.setHours(10);

// Ajouter/soustraire du temps
date.setDate(date.getDate() + 7);     // +7 jours
date.setMonth(date.getMonth() - 1);   // -1 mois
```

```
23
24 // Formater
25 date.toLocaleDateString('fr-FR');     // '25/12/2026'
26 date.toLocaleTimeString('fr-FR');     // '10:30:00'
27 date.toLocaleString('fr-FR');         // '25/12/2026 10:30:00'
28 date.toISOString();                   // '2026-12-25T09:30:00.000Z
       '
29
30 // Options de formatage
31 date.toLocaleDateString('fr-FR', {
32   weekday: 'long',
33   year: 'numeric',
34   month: 'long',
35   day: 'numeric'
36 });
37 // 'vendredi 25 decembre 2026'
```

12.3 Calculer avec les dates

```
1  const debut = new Date('2025-01-01');
2  const fin = new Date('2025-12-31');
3
4  // Difference en millisecondes
5  const diffMs = fin - debut;
6
7  // Convertir en jours
8  const diffJours = diffMs / (1000 * 60 * 60 * 24);
9  console.log(diffJours);  // 364
10
11 // Comparer des dates
12 debut < fin;    // true
13 debut > fin;    // false
14
15 // Verifier si une date est valide
16 function estDateValide(date) {
17   return date instanceof Date && !isNaN(date);
18 }
```

13 Les classes

13.1 Déclaration d'une classe

```
class Personne {
  // Constructeur
  constructor(nom, age) {
    this.nom = nom;
    this.age = age;
  }

  // Methode d'instance
  sePresenter() {
    return `Je m'appelle ${this.nom} et j'ai ${this.age} ans.`;
  }

  // Getter
  get estMajeur() {
    return this.age >= 18;
  }

  // Setter
  set nouveauNom(nom) {
    if (nom.length > 0) {
      this.nom = nom;
    }
  }
}

// Utilisation
const alice = new Personne('Alice', 25);
alice.sePresenter();       // "Je m'appelle Alice..."
alice.estMajeur;           // true
alice.nouveauNom = 'Alicia';
```

13.2 Héritage

```
class Etudiant extends Personne {
  constructor(nom, age, formation) {
    super(nom, age);       // Appel du constructeur parent
    this.formation = formation;
  }

  // Surcharge de methode
  sePresenter() {
    return `${super.sePresenter()} J'etudie ${this.formation}.`;
  }

  // Nouvelle methode
  etudier() {
```

```
    return `${this.nom} etudie...`;
  }
}

const bob = new Etudiant('Bob', 20, 'informatique');
bob.sePresenter();
// "Je m'appelle Bob et j'ai 20 ans. J'etudie informatique."
bob.etudier();            // "Bob etudie..."
bob.estMajeur;            // true (herite)
```

13.3 Membres statiques et privés

```
class Compteur {
  // Propriete statique (partagee entre toutes les instances)
  static nombreInstances = 0;

  // Champ prive (ES2022)
  #valeur = 0;

  constructor() {
    Compteur.nombreInstances++;
  }

  // Methode statique
  static getInfo() {
    return `${Compteur.nombreInstances} compteurs crees`;
  }

  // Acces au champ prive
  incrementer() {
    this.#valeur++;
  }

  get valeur() {
    return this.#valeur;
  }
}

const c1 = new Compteur();
const c2 = new Compteur();

Compteur.nombreInstances;  // 2
Compteur.getInfo();        // "2 compteurs crees"

c1.incrementer();
c1.valeur;                 // 1
// c1.#valeur;             // Erreur! Prive
```

13.4 instanceof

```
const alice = new Personne('Alice', 25);
const bob = new Etudiant('Bob', 20, 'info');

alice instanceof Personne;    // true
alice instanceof Etudiant;    // false
bob instanceof Etudiant;      // true
bob instanceof Personne;      // true (heritage)
bob instanceof Object;        // true (tout herite d'Object)

// Verifier le type
typeof alice;                 // 'object'
alice.constructor.name;       // 'Personne'
```

14 La gestion d'erreurs

14.1 try...catch...finally

```
try {
  // Code qui peut generer une erreur
  const resultat = risque();
  console.log(resultat);
} catch (error) {
  // Gestion de l'erreur
  console.error('Erreur:', error.message);
} finally {
  // Execute toujours (optionnel)
  console.log('Nettoyage...');
}

// L'objet Error
try {
  throw new Error('Quelque chose a mal tourne');
} catch (e) {
  console.log(e.name);        // 'Error'
  console.log(e.message);     // 'Quelque chose a mal tourne'
  console.log(e.stack);       // Stack trace
}
```

14.2 Types d'erreurs et erreurs personnalisées

```
// Types d'erreurs natifs
new Error('Erreur generique');
new SyntaxError('Erreur de syntaxe');
new TypeError('Erreur de type');
new ReferenceError('Variable non definie');
new RangeError('Valeur hors limites');

// Erreur personnalisee
class ValidationError extends Error {
  constructor(message, champ) {
    super(message);
    this.name = 'ValidationError';
    this.champ = champ;
  }
}

function valider(email) {
  if (!email.includes('@')) {
    throw new ValidationError('Email invalide', 'email');
  }
}

try {
```

```
    valider('invalid');
  } catch (e) {
    if (e instanceof ValidationError) {
      console.log(`Erreur sur ${e.champ}: ${e.message}`);
    }
  }
```

14.3 Débogage

```
// Point d'arret dans le code
debugger;  // Pause l'execution si DevTools ouvert

// Assertions (developpement)
console.assert(x > 0, 'x doit etre positif');

// Stack trace
console.trace('Ou suis-je?');
```

Astuce

Chrome DevTools (F12) :

- **Sources** : Points d'arrêt, pas à pas
- **Console** : Exécuter du code, inspecter
- **Network** : Requêtes HTTP
- **Elements** : DOM et CSS en temps réel

15 Résumé et points clés

À retenir

Variables et types :

- Utilisez `const` par défaut, `let` si réassignation nécessaire
- Types primitifs : string, number, boolean, null, undefined, symbol, bigint
- Utilisez `===` pour les comparaisons (égalité stricte)
- Valeurs falsy : `false`, `0`, `''`, `null`, `undefined`, `NaN`

À retenir

Fonctions :

- Fonctions fléchées : `(a, b) => a + b`
- Paramètres par défaut, opérateur rest (`...args`)
- Les callbacks sont des fonctions passées en argument
- Les closures capturent leur environnement lexical

À retenir

Objets et tableaux :

- Objets : `{ cle: valeur }`, accès via `obj.cle` ou `obj['cle']`
- Tableaux : `map`, `filter`, `reduce`, `forEach`
- Destructuring : `const { a, b } = obj` et `const [x, y] = arr`
- Spread : `...obj`, `...arr`

À retenir

DOM et événements :

- Sélection : `querySelector`, `querySelectorAll`
- Modification : `textContent`, `innerHTML`, `classList`
- Événements : `addEventListener('event', handler)`
- `event.preventDefault()` pour bloquer le comportement par défaut

À retenir

Classes et modules :

- Classes : `class`, `constructor`, `extends`, `super`
- Modules : `export`, `import`, `export default`
- Champs privés avec `#`

Prochain chapitre : TypeScript

Nous ajouterons le typage statique à JavaScript pour un code plus robuste !