

Référence rapide JavaScript

ES6+, DOM, événements, Fetch API, async/await

Bases

Variables

```
let name = "Alice"; // reassignable
const PI = 3.14; // constant
var old = "avoid"; // function-scoped (legacy)
```

Types de données

string	Texte: "hello" ou 'hello'
number	Entier ou décimal: 42, 3.14
boolean	true / false
null	Valeur vide intentionnelle
undefined	Déclaré mais non assigné
object	Paires clé-valeur: { a: 1 }
array	Liste ordonnée: [1, 2, 3]
symbol	Identifiant unique

Vérification & Conversion de type

```
typeof "hello" // "string"
typeof 42 // "number"
Number("42") // 42
String(100) // "100"
parseInt("3.9") // 3
parseFloat("3.14") // 3.14
```

Chaînes de caractères

Littéraux de gabarit

```
const name = "Alice";
`Hello, ${name}!` // Hello, Alice!
`Total: ${2 + 3}` // Total: 5
`Multi
line string`
```

Méthodes de chaîne

s.length	Nombre de caractères
s.toUpperCase()	Copie en MAJUSCULES
s.toLowerCase()	Copie en minuscules
s.trim()	Supprimer espaces en début/fin
s.split(",")	Découper en tableau
s.includes("x")	Vérifier la présence → bool
s.indexOf("x")	Premier index (-1 si absent)
s.slice(1, 4)	Sous-chaîne par index
s.replace(a, b)	Remplacer la première occurrence
s.replaceAll(a, b)	Remplacer toutes les occurrences
s.startsWith(x)	Vérifier le préfixe → bool
s.endsWith(x)	Vérifier le suffixe → bool
s.padStart(n, c)	Compléter le début jusqu'à la longueur n

Tableaux

Créer & Accéder

```
const fruits = ["apple", "banana", "cherry"];
fruits[0] // "apple"
fruits.length // 3
fruits.at(-1) // "cherry"
```

Méthodes mutantes

arr.push(x)	Ajouter en fin
arr.pop()	Retirer & retourner le dernier
arr.unshift(x)	Ajouter au début
arr.shift()	Retirer & retourner le premier
arr.splice(i, n)	Supprimer n éléments à l'index i
arr.sort()	Trier en place (lexicographique)
arr.reverse()	Inverser en place

Méthodes non-mutantes

arr.map(fn)	Transformer chaque élément
arr.filter(fn)	Garder les éléments où fn est vrai
arr.reduce(fn, init)	Accumuler en une seule valeur
arr.find(fn)	Première correspondance ou undefined
arr.findIndex(fn)	Index de la première correspondance (-1)
arr.includes(x)	Vérifier la présence → bool
arr.slice(a, b)	Copie superficielle d'une portion
arr.join(",")	Joindre en chaîne
arr.forEach(fn)	Itérer (sans valeur de retour)
[...a, ...b]	Concaténer des tableaux (spread)

Objets

Créer & Accéder

```
const user = { name: "Alice", age: 20 };
user.name // "Alice"
user["age"] // 20
user.gpa = 3.85; // add/update
```

Déstructuration & Spread

```
const { name, age } = user;
const copy = { ...user, age: 21 };
```

Méthodes d'Object

Object.keys(o)	Tableau des clés
Object.values(o)	Tableau des valeurs
Object.entries(o)	Tableau de paires [clé, valeur]
Object.assign(t, s)	Copier les propriétés de s vers t
"k" in obj	La clé existe? → bool
delete obj.k	Supprimer une propriété
Object.freeze(o)	Rendre immuable (superficiel)

Structures de contrôle

if / else if / else

```
if (score >= 90) {
  grade = "A";
} else if (score >= 80) {
  grade = "B";
} else {
  grade = "C";
}
```

Ternaire & Coalescence nulle

```
const status = score >= 60 ? "pass" : "fail";
const name = user.name ?? "Anonymous";
```

switch

```
switch (color) {
  case "red": stop(); break;
  case "green": go(); break;
  default: wait();
}
```

Boucles

for / for...of / for...in

```
for (let i = 0; i < 5; i++) { }

for (const item of ["a", "b"]) { } // arrays

for (const key in obj) { } // object keys
```

while / do...while

```
while (count < 10) { count++; }

do { count++; } while (count < 10);
```

break & continue

```
for (let i = 0; i < 10; i++) {
  if (i === 5) break; // stop loop
  if (i % 2 === 0) continue; // skip
}
```

Fonctions

Déclaration & Fléchée

```
function greet(name) {
  return `Hello, ${name}!`;
}
const greet = (name) => `Hello, ${name}!`;
const square = x => x * x; // single param
```

Paramètres par défaut & Rest

```
function greet(name = "World") { }

function sum(...nums) {
  return nums.reduce((a, b) => a + b, 0);
}
```

Callbacks

```
[1, 2, 3].map(x => x * 2); // [2, 4, 6]
[1, 2, 3].filter(x => x > 1); // [2, 3]
setTimeout(() => console.log("done"), 1000);
```

Classes

```
class Dog {
  constructor(name, breed) {
    this.name = name;
    this.breed = breed;
  }
  bark() { return `${this.name} says Woof!`; }
}

class Puppy extends Dog {
  constructor(name, breed, toy) {
    super(name, breed);
    this.toy = toy;
  }
}
```

Gestion des erreurs

```
try {
  JSON.parse("bad json");
} catch (err) {
  console.error(err.message);
} finally {
  console.log("always runs");
}
```

Lever une erreur

```
throw new Error("Something went wrong");
```

DOM

Sélectionner des éléments

```
document.querySelector(".cls") // first match
document.querySelectorAll("li") // all matches
document.getElementById("main")
```

Référence rapide JavaScript

Modifier des éléments

```
el.textContent = "new text";
el.innerHTML = "<b>bold</b>";
el.style.color = "red";
el.classList.add("active");
el.classList.toggle("hidden");
el.setAttribute("data-id", "42");
```

Événements

```
btn.addEventListener("click", (e) => {
  console.log(e.target);
});
```

Créer des éléments

```
const li = document.createElement("li");
li.textContent = "New item";
ul.appendChild(li);
el.remove(); // remove element
```

Fetch API

Requête GET

```
fetch("https://api.example.com/data")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Requête POST

```
fetch(url, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ key: "value" }),
});
```

Async / Await

```
async function loadData() {
  try {
    const res = await fetch(url);
    const data = await res.json();
    return data;
  } catch (err) {
    console.error(err);
  }
}
```

Requêtes parallèles

```
const [users, posts] = await Promise.all([
  fetch("/users").then(r => r.json()),
  fetch("/posts").then(r => r.json()),
]);
```

Modules

Exports nommés

```
// math.js
export const PI = 3.14;
export function add(a, b) { return a + b; }

// main.js
import { PI, add } from "./math.js";
```

Export par défaut

```
// logger.js
export default function log(msg) { }

// main.js
import log from "./logger.js";
```