

Referência Rápida de JavaScript

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

Básico

Variáveis

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

Tipos de Dados

string	Texto: "hello" ou 'hello'
number	Inteiro ou float: 42, 3.14
boolean	true/false
null	Valor vazio intencional
undefined	Declarado mas não atribuído
object	Pares chave-valor: { a: 1 }
array	Lista ordenada: [1, 2, 3]
symbol	Identificador único

Verificação & Conversão de Tipo

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

Strings

Template Literals

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

Métodos de String

s.length	Número de caracteres
s.toUpperCase()	Cópia em MAIÚSCULAS
s.toLowerCase()	Cópia em minúsculas
s.trim()	Remover espaços no início/fim
s.split(",")	Dividir em array
s.includes("x")	Verificar se contém → bool
s.indexOf("x")	Primeiro índice (-1 se não encontrar)
s.slice(1, 4)	Substrng por índice
s.replace(a, b)	Substituir primeira ocorrência
s.replaceAll(a, b)	Substituir todas as ocorrências
s.startsWith(x)	Verificar prefixo → bool
s.endsWith(x)	Verificar sufixo → bool
s.padStart(n, c)	Preencher início até o tamanho n

Arrays

Criar & Acessar

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

Métodos Mutantes

arr.push(x)	Adicionar ao final
arr.pop()	Remover e retornar o último
arr.unshift(x)	Adicionar ao início
arr.shift()	Remover e retornar o primeiro
arr.splice(i, n)	Remover n itens no índice i
arr.sort()	Ordenar in-place (lexicográfico)
arr.reverse()	Inverter in-place

Métodos Não-Mutantes

arr.map(fn)	Transformar cada elemento
arr.filter(fn)	Manter elementos onde fn é true
arr.reduce(fn, init)	Acumular em valor único
arr.find(fn)	Primeiro match ou undefined
arr.findIndex(fn)	Índice do primeiro match (-1)
arr.includes(x)	Verificar se contém → bool
arr.slice(a, b)	Cópia rasa de parte do array
arr.join(",")	Unir em string
arr.forEach(fn)	Iterar (sem retorno)
[...a, ...b]	Concatenar arrays (spread)

Objetos

Criar & Acessar

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

Desestruturação & Spread

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

Métodos de Object

Object.keys(o)	Array de chaves
Object.values(o)	Array de valores
Object.entries(o)	Array de pares [chave, valor]
Object.assign(t, s)	Copiar propriedades s → t
"k" in obj	Chave existe? → bool
delete obj.k	Remover propriedade
Object.freeze(o)	Tornar imutável (shallow)

Controle de Fluxo

if / else if / else

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

Ternário & Nullish Coalescing

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

switch

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

Laços

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
}
```

Funções

Declaração & Arrow Function

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

Parâmetros Padrão & 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;
  }
}
```

Tratamento de Erros

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

Lançar Erros

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

DOM

Selecionar Elementos

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

Referência Rápida de JavaScript

Modificar Elementos

```
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");
```

Eventos

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

Criar Elementos

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

Fetch API

Requisição GET

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

Requisição 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);
  }
}
```

Requisições Paralelas

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

Módulos

Exports Nomeados

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

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

Export Padrão

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

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