JAVASCRIPT QUICK REFERENCE

Basics

Variables

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

Data Types

`string`	Text: `"hello"` or `'hello'`
`number`	Integer or float: `42`, `3.14`
`boolean`	`true` / `false`
`null`	Intentional empty value
`undefined`	Declared but not assigned
`object`	Key-value pairs: `{ a: 1 }`
`array`	Ordered list: `[1, 2, 3]`
`symbol`	Unique identifier

Type Checking & Conversion

```
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`
```

String Methods

`s.length`	Number of characters
`s.toUpperCase()`	UPPERCASE copy
`s.toLowerCase()`	lowercase copy
`s.trim()`	Remove leading/trailing whitespace
`s.split(",")`	Split into array
`s.includes("x")`	Contains check → bool
`s.indexOf("x")`	First index (-1 if none)
`s.slice(1, 4)`	Substring by index
`s.replace(a, b)`	Replace first match
`s.replaceAll(a, b)`	Replace all matches
`s.startsWith(x)`	Check prefix → bool
`s.endsWith(x)`	Check suffix → bool
`s.padStart(n, c)`	Pad start to length n

Arrays

Creating & Accessing

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

Mutating Methods

`arr.push(x)`	Add to end
`arr.pop()`	Remove & return last
`arr.unshift(x)`	Add to start
`arr.shift()`	Remove & return first
`arr.splice(i, n)`	Remove n items at index i
`arr.sort()`	Sort in place (lexicographic)
`arr.reverse()`	Reverse in place

Non-Mutating Methods

`arr.map(fn)`	Transform each element
`arr.filter(fn)`	Keep elements where fn is true
`arr.reduce(fn, init)`	Accumulate into single value
`arr.find(fn)`	First match or undefined
`arr.findIndex(fn)`	Index of first match (-1)
`arr.includes(x)`	Contains check → bool
`arr.slice(a, b)`	Shallow copy of portion
`arr.join(",")`	Join into string
`arr.forEach(fn)`	Iterate (no return value)
`[...a, ...b]`	Concatenate arrays (spread)

Objects

Creating & Accessing

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

Destructuring & Spread

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

Object Methods

`Object.keys(o)`	Array of keys
`Object.values(o)`	Array of values
`Object.entries(o)`	Array of [key, value] pairs
`Object.assign(t, s)`	Copy properties s → t
`"k" in obj`	Key exists? → bool
`delete obj.k`	Remove property
`Object.freeze(o)`	Make immutable (shallow)

Control Flow

if / else if / else

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

Ternary & 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();
}
```

Loops

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
}
```

Functions

Function Declaration & Arrow

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

Default Parameters & 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;
  }
}
```

Error Handling

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

Throwing Errors

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

JAVASCRIPT QUICK REFERENCE (continued)

DOM

Selecting Elements

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

Modifying Elements

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

Events

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

Creating Elements

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

Fetch API

GET Request

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

POST Request

```
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);
  }
}
```

Parallel Requests

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

Modules

Named Exports

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

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

Default Export

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

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