JAVASCRIPT

ES6, DOM and Async, explained simply

JavaScript Notes · easy to understand · free for students

I. Start with JavaScript

- JavaScript (JS) is the programming language of the web. It makes pages interactive: button clicks, form checks, menus, animations, live data.
- It runs in the browser (and also on servers with Node.js). Every browser has a built-in JS engine, so you only need a text editor.

IDEA Think of a human body. HTML is the skeleton, CSS is the clothes and style, and JavaScript is the movement.

Language	Job	Example
HTML	Content and structure	Heading, button, image
CSS	Design and layout	Color, spacing, flexbox
JavaScript	Behaviour and logic	Click, fetch data, validate

EXAMPLE Add JS to a page. Save as index.html and open it in the browser.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JS Demo</title>
  <script src="app.js" defer></script>   <!-- external file, runs after HTML loads -->
</head>
<body>
  <h1 id="title">Hello</h1>
  <button id="btn">Click me</button>
</body>
</html>
```

EXAMPLE app.js: your first script.

```
console.log("Hello from JavaScript");        // prints in the browser console
document.getElementById("btn").addEventListener("click", () => {
  alert("Button clicked!");
});
```

- Press F12 in the browser and open the Console tab to see console.log output and errors. Use defer on the script tag so the HTML is ready before your code runs.

2. Variables and Data Types

- A variable stores a value. Use let for values that change and const for values that do not. Avoid var in new code.

Keyword	Scope	Reassign	Notes
let	Block { }	Yes	Modern choice for changing values
const	Block { }	No	Default choice. Objects inside can still change
var	Function	Yes	Old style, hoisting surprises. Avoid

Type	Example	Note
number	42, 3.14, NaN	One type for integers and decimals
string	'hi', "hi", hi	Text
boolean	true, false	Yes or no
undefined	let x;	Declared but no value
null	let y = null;	Empty on purpose
object	{ }, [ ], new Date()	Arrays are objects too
bigint, symbol	10n, Symbol()	Rare, for special cases

EXAMPLE Variables, typeof and template strings.

```
let age = 20;
const name = "Asha";
age = 21;                                  // allowed with let
// name = "Ravi";                          // error: const cannot be reassigned

console.log(typeof age);                   // "number"
console.log(typeof name);                  // "string"
console.log(`Hi ${name}, you are ${age}`);   // template literal
console.log(10 / 3, 10 % 3, 2 ** 3);       // 3.333  1  8
```

Strings and type conversion

EXAMPLE *Common string methods and conversion.*

```
const s = "  Hello World  ";
s.trim();                  // "Hello World"
s.toUpperCase();           // "  HELLO WORLD  "
s.includes("World");       // true
s.slice(2, 7);             // "Hello"
s.split(" ");              // ["", "", "Hello", "World", "", ""]
"a-b-c".replace("-", "+"); // "a+b-c"

Number("42");              // 42
parseInt("42px");          // 42
String(99);                // "99"
Boolean("");               // false
```

REMEMBER *Always use `===` (strict equal) and not `==`. `5 == "5"` is true but `5 === "5"` is false. Falsy values are `false`, `0`, `""`, `null`, `undefined`, `NaN`. Everything else is truthy.*

3. Operators and Control Flow

Kind	Operators	Example
Arithmetic	`+ - * / % **`	`7 % 2`
Compare	`=== !== < > <= >=`	`a >= b`
Logical	`&& || !`	`a > 1 && b < 5`
Assign	`= += -= *= /= ++`	`count++`
Nullish	`?? ?. ??=`	`user?.name ?? "Guest"`

EXAMPLE Decisions.

```
const marks = 72;
if (marks >= 90) {
  console.log("A");
} else if (marks >= 60) {
  console.log("B");
} else {
  console.log("C");
}

const result = marks >= 40 ? "Pass" : "Fail";   // ternary

switch (new Date().getDay()) {
  case 0: console.log("Sunday"); break;
  case 6: console.log("Saturday"); break;
  default: console.log("Weekday");
}
```

EXAMPLE Loops.

```
for (let i = 0; i < 3; i++) console.log(i);        // 0 1 2

let n = 3;
while (n > 0) { console.log(n); n--; }             // 3 2 1

for (const fruit of ["apple", "mango"]) {          // for...of: values of an array
  console.log(fruit);
}
for (const key in { a: 1, b: 2 }) {                // for...in: keys of an object
  console.log(key);
}
```

- `break` leaves a loop, `continue` skips to the next turn, `return` leaves a function.

4. Functions

- A function is a reusable block of code. It can take parameters and give back a return value.

EXAMPLE *Three ways to write a function.*

```
function add(a, b) { return a + b; }            // declaration (hoisted)

const multiply = function (a, b) { return a * b; };   // expression

const square = (x) => x * x;                    // arrow function
const greet = (name = "Guest") => `Hello ${name}`;    // default parameter

console.log(add(2, 3), square(4), greet());     // 5  16  Hello Guest
```

➤ Rest, spread and callbacks

EXAMPLE *Rest parameters, spread and a callback.*

```
const sum = (...nums) => nums.reduce((t, n) => t + n, 0);   // rest: collect args
console.log(sum(1, 2, 3, 4));                               // 10

const a = [1, 2], b = [3, 4];
const all = [...a, ...b];                                   // spread: expand
console.log(all);                                           // [1, 2, 3, 4]

function run(task) { task(); }                              // takes a function
run(() => console.log("I am a callback"));
```

➤ Scope and closure

- *Scope decides where a variable can be used. A closure is an inner function that remembers the variables of its outer function, even after the outer function ends.*

EXAMPLE *Closure: a private counter.*

```
function makeCounter() {
  let count = 0;                // hidden from outside
  return () => ++count;         // inner function remembers count
}
const next = makeCounter();
console.log(next(), next(), next());     // 1  2  3
```

REMEMBER *Arrow functions have no own `this`. They use `this` from the place where they were written. Use normal functions for object methods that need `this`.*

5. Arrays and Objects

- *An array is an ordered list. An object stores data as key: value pairs. Most real data in JS is an array of objects.*

Method	What it does	Returns
`push / pop`	Add / remove at the end	new length / item
`unshift / shift`	Add / remove at the start	new length / item
`map(fn)`	Change every item	New array
`filter(fn)`	Keep items that pass a test	New array
`reduce(fn, start)`	Combine all items into one value	One value
`find(fn)`	First item that passes	Item or undefined
`includes(x)`	Is x in the array?	true / false
`sort(fn)`	Sort in place (give a compare function for numbers)	Same array
`slice / splice`	Copy a part / remove or insert items	Part / removed items

EXAMPLE map, filter, reduce, find.

```
const nums = [5, 2, 9, 1];
const doubled = nums.map(n => n * 2);              // [10, 4, 18, 2]
const big = nums.filter(n => n > 3);               // [5, 9]
const total = nums.reduce((t, n) => t + n, 0);     // 17
const firstBig = nums.find(n => n > 6);            // 9
const sorted = [...nums].sort((a, b) => a - b);    // [1, 2, 5, 9]

nums.forEach((n, i) => console.log(i, n));         // loop, returns nothing
```

Objects

EXAMPLE Create, read, change and loop over an object.

```
const user = {
  name: "Asha",
  age: 20,
  skills: ["HTML", "JS"],
  greet() { return `Hi, I am ${this.name}`; }      // method
};

console.log(user.name, user["age"]);               // dot and bracket access
user.city = "Delhi";                               // add a property
delete user.age;                                   // remove a property
console.log(user.greet());                         // Hi, I am Asha

console.log(Object.keys(user));                    // ["name","skills","greet","city"]
for (const [key, value] of Object.entries(user)) console.log(key, value);
```

JSON

- JSON is a text format for data. Use `JSON.stringify` to turn an object into text and `JSON.parse` to turn text back into an object.

EXAMPLE JSON round trip.

```
const text = JSON.stringify({ id: 1, name: "Asha" });    // '{"id":1,"name":"Asha"}'
const obj = JSON.parse(text);                             // { id: 1, name: "Asha" }
console.log(obj.name);
```

6. ES6 Features

- ES6 (ECMAScript 2015) and later versions added many features that make code shorter and cleaner. All modern browsers support them.

Feature	Example	Why use it
let / const	const x = 1;	Safer variables
Arrow function	(a, b) => a + b	Short functions
Template literal	Hi ${name}	Easy strings
Destructuring	const { a, b } = obj;	Pull values out
Spread / rest	[...arr], (...args)	Copy, merge, collect
Default parameter	function f(x = 1)	Safe defaults
Optional chaining	user?.address?.city	No crash on missing data
Nullish ??	value ?? 'none'	Default only for null/undefined
Class	class Car { }	Blueprint for objects
Modules	import / export	Split code in files
Map, Set	new Map(), new Set()	Better key-value and unique lists

➤ Destructuring, spread and optional chaining

EXAMPLE Take apart arrays and objects.

```
const [first, second] = ["red", "green", "blue"];      // array destructuring
const { name, age = 18 } = { name: "Ravi" };           // object, with default
console.log(first, name, age);                         // red Ravi 18

const user = { name: "Asha", address: { city: "Pune" } };
const copy = { ...user, name: "Meera" };               // copy and change one key
console.log(user.address?.city);                       // "Pune"
console.log(user.job?.title ?? "No job");              // "No job" (no error)
```

Classes

EXAMPLE Class with inheritance.

```
class Animal {
  #secret = 42;                          // private field
  constructor(name) { this.name = name; }
  speak() { return `${this.name} makes a sound`; }
}
class Dog extends Animal {
  speak() { return `${this.name} barks`; }   // override
}
const d = new Dog("Tommy");
console.log(d.speak());                  // Tommy barks
```

Modules, Map and Set

EXAMPLE Split code into files.

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

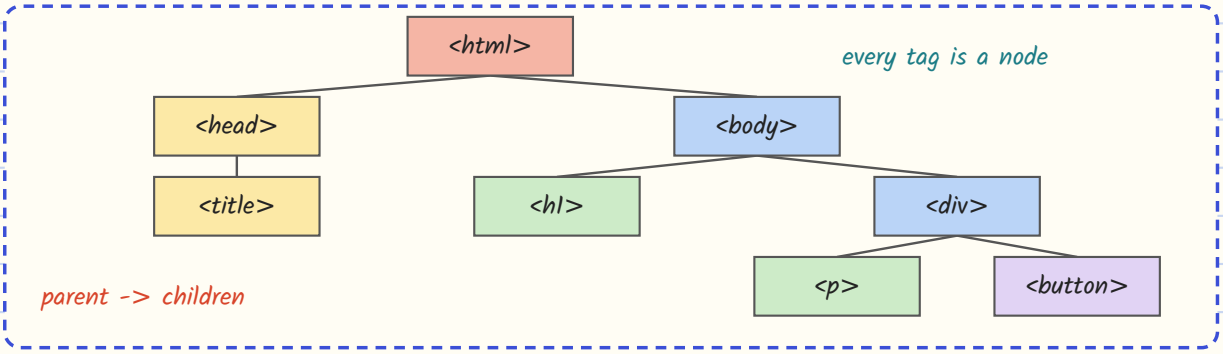
// main.js
import add, { PI } from "./math.js";
console.log(add(2, 3), PI);

// in HTML:  <script type="module" src="main.js"></script>

const set = new Set([1, 1, 2]);          // {1, 2}  only unique values
const map = new Map([["a", 1]]);         // keys can be any type
map.set("b", 2); console.log(map.get("b"));
```

7. The DOM

- The DOM (Document Object Model) is the page turned into a tree of objects. JavaScript uses it to read and change the page. The main object is `document`.



Select elements

Method	Finds
`document.getElementById('id')`	One element by id
`document.querySelector('.card')`	First element that matches a CSS selector
`document.querySelectorAll('li')`	All matches (a NodeList, use forEach)
`element.children, parentElement`	Move up and down the tree

EXAMPLE Read and change the page.

```
const title = document.querySelector("#title");
title.textContent = "New heading";              // change text (safe)
title.style.color = "tomato";                   // inline style
title.classList.add("big");                     // add a CSS class
title.classList.toggle("active");               // add if missing, remove if present
title.setAttribute("title", "Tooltip text");

const items = document.querySelectorAll("li");
items.forEach(li => li.classList.add("item"));
```

EXAMPLE Create and remove elements.

```
const ul = document.querySelector("ul");
const li = document.createElement("li");        // 1. create
li.textContent = "New item";                    // 2. fill
ul.append(li);                                  // 3. add to the page

li.remove();                                    // remove it
// innerHTML parses HTML: never put user input in it (XSS risk). Use textContent.
```

REMEMBER `textContent` sets plain text and is safe. `innerHTML` reads HTML tags and can run bad code if you put user text in it.

8. Events

- An event is something that happens: a click, a key press, a form submit. You react with an event listener.

Event	When it happens
`click`	Mouse click or tap
`input, change`	Input value changes while typing / after it finishes
`submit`	A form is submitted
`keydown, keyup`	Keyboard key pressed or released
`mouseover, mouseout`	Mouse enters or leaves
`DOMContentLoaded`	HTML is fully loaded
`scroll, resize`	Page scrolls, window size changes

EXAMPLE Click, input and form submit.

```
const btn = document.querySelector("#btn");
btn.addEventListener("click", (event) => {
  console.log("Clicked", event.target);          // event.target = the element clicked
});

document.querySelector("#name").addEventListener("input", (e) => {
  console.log(e.target.value);                   // live text of the input
});

document.querySelector("form").addEventListener("submit", (e) => {
  e.preventDefault();                            // stop the page from reloading
  console.log("Form handled by JS");
});
```

➤ Bubbling and event delegation

- Events bubble up: a click on a child also reaches its parent. So you can put one listener on the parent to handle many children. This is event delegation and it also works for items added later.

EXAMPLE One listener for a whole list.

```
document.querySelector("ul").addEventListener("click", (e) => {
  const li = e.target.closest("li");             // which item was clicked?
  if (li) li.classList.toggle("done");
});
```

EXAMPLE Mini project: a counter.

```
<button id="minus">-</button> <span id="count">0</span> <button id="plus">+</button>

<script>
  let count = 0;
  const show = document.querySelector("#count");
  document.querySelector("#plus").addEventListener("click", () => show.textContent = ++count);
  document.querySelector("#minus").addEventListener("click", () => show.textContent = --count);
</script>
```

9. Asynchronous JavaScript

- JS runs one thing at a time (single threaded). Slow work like timers, network calls and files must not block the page, so it is done asynchronously: start it now, get the result later.

IDEA In a restaurant the waiter takes your order, gives it to the kitchen and serves other tables while the food cooks. He does not stand and wait. That is async.

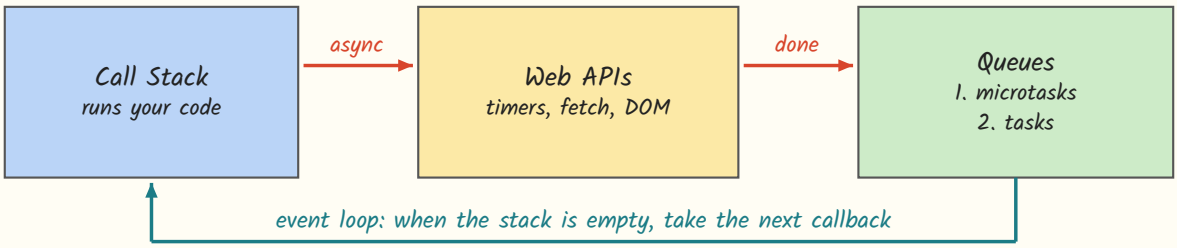
EXAMPLE setTimeout and the callback style.

```
console.log("1. Start");
setTimeout(() => console.log("2. After 1 second"), 1000);   // runs later
console.log("3. End");
// output: 1. Start, 3. End, 2. After 1 second

// callback hell: callbacks inside callbacks are hard to read
getUser(1, (user) => {
  getOrders(user.id, (orders) => {
    getItems(orders[0], (items) => console.log(items));
  });
});
```

The event loop

- The call stack runs your code. Slow tasks go to the Web APIs (browser). When they finish, their callback waits in a queue. The event loop moves a callback to the stack only when the stack is empty.

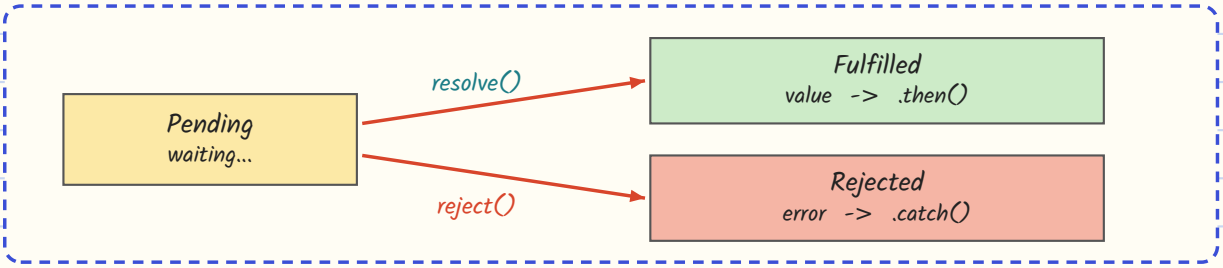


EXAMPLE Order of output: promises run before timers.

```
console.log(1);
setTimeout(() => console.log(2), 0);          // task queue
Promise.resolve().then(() => console.log(3)); // microtask queue (higher priority)
console.log(4);
// output: 1, 4, 3, 2
```

10. Promises

- A Promise is an object that stands for a value that will come later. It is in one of three states: pending, fulfilled (success) or rejected (error).



EXAMPLE Create and use a promise.

```
const wait = (ms) => new Promise((resolve, reject) => {
  if (ms < 0) reject(new Error("Negative time"));
  else setTimeout(() => resolve(`Waited ${ms} ms`), ms);
});

wait(500)
  .then((msg) => { console.log(msg); return wait(300); })   // chain
  .then((msg) => console.log(msg))
  .catch((err) => console.error("Failed:", err.message))    // any error above
  .finally(() => console.log("Done"));                      // always runs
```

Method	Meaning
`Promise.all([p1, p2])`	Wait for all. Fails if any one fails
`Promise.allSettled([...])`	Wait for all, give each success or error
`Promise.race([...])`	First one to finish wins
`Promise.any([...])`	First one to succeed wins

II. Async / Await

- async / await is a cleaner way to use promises. It looks like normal top-to-bottom code. `await` pauses inside the async function until the promise is ready.

EXAMPLE The same flow, easier to read.

```
const wait = (ms) => new Promise((r) => setTimeout(r, ms));

async function run() {
  try {
    console.log("Start");
    await wait(1000);                // pauses here, page keeps working
    console.log("After 1 second");
    return "finished";               // an async function always returns a promise
  } catch (err) {
    console.error(err);
  } finally {
    console.log("Cleanup");
  }
}
run().then(console.log);
```

Run tasks together

EXAMPLE *Sequential is slow, parallel is fast.*

```
async function slow() {
  const a = await wait(1000);
  const b = await wait(1000);         // starts only after a: total 2 seconds
}
async function fast() {
  const [a, b] = await Promise.all([wait(1000), wait(1000)]);   // total 1 second
}
```

REMEMBER `await` *works only inside an* `async` *function (or at the top of a module). Always wrap* `await` *in* `try / catch` *to handle errors.*

12. Fetch API and Storage

- *fetch gets data from a server (an API). It returns a promise. The response has* `.json()` *to read the data, which is also a promise.*

EXAMPLE *GET request.*

```
async function loadUsers() {
  try {
    const res = await fetch("https://jsonplaceholder.typicode.com/users");
    if (!res.ok) throw new Error("HTTP error " + res.status);   // fetch does not fail on 404
    const users = await res.json();                             // parse JSON
    users.slice(0, 3).forEach(u => console.log(u.name));
  } catch (err) {
    console.error("Could not load:", err.message);
  }
}
loadUsers();
```

EXAMPLE *POST request and showing data on the page.*

```
const res = await fetch("https://jsonplaceholder.typicode.com/posts", {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ title: "Hello", body: "My post", userId: 1 })
});
const data = await res.json();
console.log(data.id);

document.querySelector("#out").textContent = `Created post ${data.id}`;
```

REMEMBER `fetch` *only rejects on network failure. A 404 or 500 still counts as success, so always check* `res.ok`.

localStorage

- `localStorage` *keeps small data in the browser even after you close it. It stores only strings, so use JSON for objects.*

EXAMPLE *Save and read data.*

```
localStorage.setItem("theme", "dark");
console.log(localStorage.getItem("theme"));       // "dark"

localStorage.setItem("user", JSON.stringify({ name: "Asha" }));
const user = JSON.parse(localStorage.getItem("user"));
console.log(user.name);

localStorage.removeItem("theme");
localStorage.clear();                             // remove everything
```

13. Errors and Debugging

Error	Meaning	Common cause
ReferenceError	Variable not defined	Typo or wrong scope
TypeError	Wrong type of value	Calling undefined as a function, reading a key of null
SyntaxError	Code is not valid JS	Missing bracket or quote
RangeError	Value out of range	Bad array length, endless recursion

EXAMPLE try / catch and throw.

```
try {
  const user = null;
  console.log(user.name);                 // TypeError
} catch (err) {
  console.error(err.name, err.message);
} finally {
  console.log("Always runs");
}

function divide(a, b) {
  if (b === 0) throw new Error("Cannot divide by zero");
  return a / b;
}
```

- Debug with `console.log`, `console.table`, the *debugger* statement and the Sources tab in DevTools where you can set breakpoints.

14. Projects and Practice

Project	What you practice
1. Counter and color changer	Variables, DOM, click events
2. To-do list	Arrays, create elements, event delegation, localStorage
3. Form validator	Events, preventDefault, conditions
4. Quiz app	Arrays of objects, loops, score logic
5. Weather or user app	fetch, async/await, JSON, error handling
6. Stopwatch or timer	setInterval, functions, state
7. Image gallery with modal	Selecting many elements, classList, events

Common mistakes

Mistake	Fix
Using == to compare	Use ===
Script runs before the HTML exists	Use defer or put the script at the end of body
Forgetting await	Add await, or the value is a pending Promise
Not checking res.ok after fetch	Check res.ok and throw an error
Sorting numbers with sort()	Pass (a, b) => a - b
Using innerHTML with user text	Use textContent
Changing a const array and thinking it is locked	const stops reassign, not push or change

Practice questions

1. What is the difference between var, let and const?
2. What is the difference between == and ===? What are falsy values?
3. What is a closure? Give a small example.
4. Explain map, filter and reduce with examples.
5. What is destructuring? What are spread and rest?
6. How does the DOM work? How do you select and change an element?
7. What is event bubbling and event delegation?
8. Explain the event loop. Why does a promise run before setTimeout?
9. What are the three states of a Promise? How do you handle errors?
10. Write an async function that fetches data and handles errors.

- Build one small project every week. Open DevTools while you code and read every error message, it tells you the line number.