Artificial Intelligence (AI) is reshaping how we build applications. JavaScript—already the language of the web—is increasingly becoming the glue that connects AI models, APIs, and interactive user experiences.

But if you're building AI-powered projects with JavaScript, you'll run into dozens of "how do I..." questions every day. That's why I've put together this guide: **100 of the most common JavaScript questions developers ask when working with AI, complete with explanations and runnable code snippets.**

Why This Guide?

When building with AI, you need to combine:

- **Core JavaScript skills** (async, closures, arrays, objects)
- **AI-focused utilities** (token handling, embeddings, vector math, retries, streaming APIs)
- **Practical patterns** (prompt cleaning, JSON validation, RAG pipelines, secure key management)

This guide is designed to be:

- **Hands-on:** All examples are runnable code.
- **Organized:** Questions are grouped into categories.
- **Practical:** Focused on real-world AI development, not just theory.

Categories Covered

Here are the key areas we tackle:

1. **Language & Types** – Handling `null`, `BigInt`, deep cloning, and immutability.
2. **Strings & Text** – Cleaning, chunking, and sanitizing prompts.
3. **Functions & Scope** – Closures, debounce/throttle, and pipelines.
4. **Arrays & Objects** – Grouping, deduping, and dynamic access.
5. **Math for AI (Vectors)** – Cosine similarity, softmax, normalization.
6. **Async, Promises & Fetch** – Retries, timeouts, streaming, and concurrency control.
7. **Parsing, Validation & JSON** – Safe parsing, schema checks, and repair strategies.

8. **Files, Blobs & Encoding** – Base64, hashing, and CSV/JSON handling.
9. **Security & Safety** – Preventing injection, storing API keys, sanitizing output.
10. **UI & Streaming UX** – Typing indicators, scrolling, and rendering AI output.
11. **Node.js & Servers** – Proxying AI APIs, storing chat history, SSE.
12. **Tools & Structured Output** – Function calling, JSON schemas, and state handling.
13. **Retrieval & RAG** – Chunking, embeddings, top-K search, context management.

A. Language & Types (1–10)

1) How do I check a variable's real type (including arrays and null)?
Use `typeof` + `Array.isArray()` + null check.

```
const t = v =>
  v === null ? 'null' :
  Array.isArray(v) ? 'array' :
  typeof v;

console.log(t(null), t([]), t({}), t(42), t('hi'));
```

2) What's the difference between `let`, `const`, and `var`?
`const` for values that won't be reassigned; `let` for block-scoped variables; avoid `var`.

```
const apiKey = 'abc';
let count = 0;
for (let i = 0; i < 3; i++) count++;
```

3) How do I clone objects/arrays safely?
Use structured clone for deep copies (browser/Node 17+), or

`JSON.parse(JSON.stringify())` for JSON-safe data.

```
const deep = structuredClone({ a: [1,2], b: { c: 3 } });
```

4) When should I use `BigInt` in AI code?
For token counts/ids exceeding 2^53-1.

```
const big = 9007199254740993n;
console.log(big + 2n);
```

5) What's the best way to compare floating numbers (embeddings, scores)?
Use an epsilon.

```
const almostEqual = (a,b,eps=1e-9) => Math.abs(a-b) < eps;
```

6) How do I freeze config objects (e.g., model options)?
Use `Object.freeze` to prevent mutation.

```
const config = Object.freeze({ model: 'my-model', temperature: 0.2 });
```

7) What's the difference between == and ===?
Use === to avoid coercion surprises (critical for safety logic).

```
console.log(0 == false, 0 === false); // true, false
```

8) How do I handle `undefined` vs `null`?
Prefer `undefined` for "missing," `null` for "intentionally empty."

```
function maybe(x) { return x ?? 'fallback'; }
```

9) How do I access deeply nested values safely?
Optional chaining and nullish coalescing.

```
const city = user?.profile?.address?.city ?? 'Unknown';
```

10) How do I ensure immutability in transformations?
Use spread and map/filter/reduce.

```
const next = { ...state, messages: [...state.messages, msg] };
```

B. Strings & Text (11–18)

11) How do I build multi-line prompts?
Template literals keep prompts readable.

```
const prompt = `You are a tutor.
Explain closures simply.
Use code and bullets.`;
```

12) How do I remove extra whitespace in prompts?
Trim and collapse.

```
const clean = s => s.trim().replace(/\s+/g, ' ');
```

13) How do I safely interpolate user input into prompts?
Sanitize and delimit to prevent prompt injection.

```
const safe = input.replace(/[<{}`$]/g, ''); // simple example
const prompt = `User said: """${safe}"""`;
```

14) How do I count characters or code points (emoji)?
Use `Array.from(str).length` for code points.

```
console.log('🙂'.length, Array.from('🙂').length); // 2, 1
```

15) How do I chunk long text for token limits?
Split on paragraph/sentence boundaries, then by size.

```
function chunk(s, n=2000) {
  const out=[]; for (let i=0;i<s.length;i+=n)
out.push(s.slice(i,i+n));
  return out;
}
```

16) How do I dedent template strings?
Small helper.

```
const dedent = s => s.replace(/\n\s+/g, '\n').trim();
```

17) How do I strip Markdown before JSON parsing?
Basic removal of fences.

```
const stripFences = s => s.replace(/```[\s\S]*?```/g, '').trim();
```

18) How do I safely build regex from user text?
Escape special chars.

```
const esc = s => s.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
```

C. Functions & Scope (19–25)

19) What's a closure and why does AI tooling use it?
Closures keep config/state (like model name) available.

```
const makeCaller = model => prompt => callModel({ model, prompt });
```

20) How do I debounce rapid input (chat box)?
Reduce API calls while typing.

```
function debounce(fn, wait=300){
  let t; return (...a)=>{ clearTimeout(t);
```

```
t=setTimeout(()=>fn(...a),wait); };
}
```

21) How do I throttle streaming UI updates?
Limit frequency of renders.

```
function throttle(fn, wait=50){
  let last=0; return (...a)=>{ const now=Date.now();
    if (now-last>wait){ last=now; fn(...a); }
  };
}
```

22) How do I memoize expensive computations (e.g., tokenization)?
Cache by key.

```
const memo = (fn) => {
  const m=new Map();
  return x => m.has(x) ? m.get(x) : (m.set(x, fn(x)), m.get(x));
};
```

23) How do I compose functions for pipelines (preprocess→call→postprocess)?

```
const pipe = (...fns) => x => fns.reduce((v,f)=>f(v), x);
```

24) What's the difference between pure and impure fns in AI code?
Pure = deterministic/no side effects; prefer for reliability and tests.

25) How do I capture this without losing context?
Use arrow functions or bind.

```
class UI { constructor(){ this.count=0; } inc=()=>{ this.count++; } }
```

D. Arrays & Objects (26–33)

26) How do I map responses to view models?

```
const vm = responses.map(r => ({ id:r.id, text:r.output, t:r.timing
}));
```

27) How do I flatten nested arrays (batched tokens)?

```
const flat = nested.flat(2);
```

28) How do I group messages by conversation id?

```
const byConv = msgs.reduce((m,x)=>{
  (m[x.convId] ||= []).push(x); return m;
}, {});
```

29) How do I sort by score (similarity)?

```
const top = docs.sort((a,b)=>b.score-a.score).slice(0,10);
```

30) How do I unique by id (dedupe results)?

```
const uniq = Object.values(docs.reduce((m,d)=> (m[d.id]=d,m), {}));
```

31) How do I merge defaults with overrides (config)?

```
const final = { ...defaults, ...overrides };
```

32) How do I validate object shapes quickly?
Minimal guard.

```
function hasFields(o, keys){ return o && keys.every(k=>k in o); }
```

33) How do I safely access dynamic keys?
Use optional chaining + defaults.

```
const val = row?.[colName] ?? '';
```

E. Math for AI (Vectors) (34–40)

34) How do I compute dot product?

```
const dot = (a,b)=>a.reduce((s,v,i)=>s+v*b[i],0);
```

35) How do I compute L2 norm?

```
const norm = a => Math.hypot(...a);
```

36) How do I compute cosine similarity (embeddings)?

```
const cosine = (a,b)=> dot(a,b)/(norm(a)*norm(b));
```

37) How do I normalize an embedding?

```
const normalize = a => { const n=norm(a); return a.map(x=>x/n); };
```

38) How do I top-K search with cosine?

```
function topK(query, items, k=5){
  return items
    .map(it => ({ it, score: cosine(query, it.vec) }))
    .sort((a,b)=>b.score-a.score)
    .slice(0,k);
}
```

39) How do I compute softmax for logits?

```
const softmax = xs => {
  const m=Math.max(...xs), ex=xs.map(x=>Math.exp(x-m));
  const s=ex.reduce((a,b)=>a+b,0);
  return ex.map(x=>x/s);
};
```

40) How do I avoid NaNs with tiny denominators?
Clamp denominators.

```
const safeDiv = (a,b,eps=1e-12)=> a/(Math.abs(b)<eps? eps : b);
```

F. Async, Promises & Fetch (41–54)

41) How do I wrap `fetch` with JSON and errors?

```
async function jfetch(url, opts){
  const res = await fetch(url, { ...opts, headers:{
'content-type':'application/json', ...(opts?.headers||{}) }});
  if(!res.ok) throw new Error(`HTTP ${res.status}`);
  return res.json();
}
```

42) How do I add retries with exponential backoff (429/5xx)?

```
async function retry(fn, tries=5, base=200){
  let e;
  for(let i=0;i<tries;i++){
    try { return await fn(); }
```

```
    catch(err){ e=err; await new Promise(r=>setTimeout(r, base*2**i));
}
  }
  throw e;
}
```

43) How do I run requests with a concurrency limit?

```
async function pLimit(limit, tasks){
  const pool=[]; const out=[];
  for(const t of tasks){
    const p=(async()=>out.push(await t()))();
    pool.push(p); if(pool.length>=limit) await
Promise.race(pool).then(()=>pool.splice(pool.findIndex(x=>x===p),1));
  }
  await Promise.all(pool); return out;
}
```

44) How do I handle timeouts on fetch?

```
async function fetchWithTimeout(url, ms=10000, opts={}){
  const ctrl=new AbortController(); const
id=setTimeout(()=>ctrl.abort(), ms);
  try{ return await fetch(url, { ...opts, signal: ctrl.signal }); }
  finally{ clearTimeout(id); }
}
```

45) How do I stream responses (SSE-like text/events)?

```
const res = await fetch('/stream');
const reader = res.body.getReader();
let decoder = new TextDecoder();
while(true){
```

```
  const {value, done} = await reader.read();
  if(done) break;
  const chunk = decoder.decode(value, { stream:true });
  // append to UI
}
```

46) How do I batch requests to respect rate limits?
Use small batches + delay.

```
for (let i=0; i<items.length; i+=10){
  const batch = items.slice(i,i+10).map(callAPI);
  await Promise.all(batch);
  await new Promise(r=>setTimeout(r, 200));
}
```

47) How do I detect and handle 429 rate limits?
Read `Retry-After`, backoff, retry.

```
if(res.status===429){ const wait =
+res.headers.get('Retry-After')*1000||1000; await delay(wait); ... }
```

48) How do I cancel an in-flight request (user stops generation)?
AbortController.

```
const ctrl = new AbortController();
const p = fetch(url, { signal: ctrl.signal });
ctrl.abort(); // later
```

49) How do I turn callback APIs into Promises?
Promisify.

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

50) How do I sequence async steps with shared state?

Use simple pipeline.

```
const result = await step3(await step2(await step1(input)));
```

51) How do I retry only on certain errors?
Predicate.

```
const shouldRetry = e => /HTTP (429|5\d{2})/.test(e.message);
```

52) How do I fan-out then fan-in (map→all)?

```
const outputs = await Promise.all(inputs.map(runTask));
```

53) How do I guard against unhandled rejections?
Always `try/catch` top-level awaits, add `process.on('unhandledRejection', ...)` in Node.

54) How do I serialize async work queues?
Use a promise chain.

```
let chain = Promise.resolve();
function enqueue(task){ chain = chain.then(task); return chain; }
```

G. Parsing, Validation & JSON (55–62)

55) How do I ensure the model returns valid JSON?
Ask for fenced JSON + validate/repair.

```
function tryParseJSON(s){ try{ return JSON.parse(s); } catch{ return null; } }
```

56) How do I extract JSON from mixed text?
Find first `{...}` block.

```
const jsonInText = s => {
  const i=s.indexOf('{'); const j=s.lastIndexOf('}');
  if(i===-1||j===-1) return null;
  try{ return JSON.parse(s.slice(i, j+1)); }catch{ return null; }
};
```

57) How do I enforce schema (lightweight)?
Write a validator.

```
function validate(obj){ return obj && typeof obj.title==='string' &&
Array.isArray(obj.items); }
```

58) How do I coerce numbers/booleans from strings?

```
const toBool = s => /^(true|1|yes)$/i.test(String(s));
const toNum  = s => Number(s);
```

59) How do I safely stringify big objects (circular)?
Use a replacer with a WeakSet.

```
const seen=new WeakSet();
const safe = JSON.stringify(obj,(k,v)=> typeof v==='object'&&v?
(seen.has(v)? '[Circular]' : (seen.add(v),v)) : v);
```

60) How do I pretty-print JSON for logs?

```
console.log(JSON.stringify(obj, null, 2));
```

61) How do I merge partial model outputs into state?
Shallow merge per key.

```
state = { ...state, ...partial };
```

62) How do I canonicalize keys (lowercase) before indexing?

```
const lc =
Object.fromEntries(Object.entries(obj).map(([k,v])=>[k.toLowerCase(),
v]));
```

H. Files, Blobs & Encoding (63–70)

63) How do I read a local file in the browser (for RAG)?

```
const input = document.querySelector('input[type=file]');
input.onchange = async () => {
  const text = await input.files[0].text();
};
```

64) How do I base64-encode binary (images to API)?

```
async function toBase64(file){
  const buf = await file.arrayBuffer();
  return btoa(String.fromCharCode(...new Uint8Array(buf)));
}
```

65) How do I stream a file upload with fetch?
Use `FormData`.

```
const fd = new FormData(); fd.append('file', file);
await fetch('/upload', { method:'POST', body: fd });
```

66) How do I parse CSV to text chunks?

```
const rows = csv.trim().split('\n').map(r=>r.split(','));
```

67) How do I hash content for dedupe?

```
async function hashText(s){
  const d = await crypto.subtle.digest('SHA-256', new
TextEncoder().encode(s));
  return Array.from(new
Uint8Array(d)).map(x=>x.toString(16).padStart(2,'0')).join('');
}
```

68) How do I save a JSON file from the browser?

```
const blob = new Blob([JSON.stringify(data,null,2)],
{type:'application/json'});
const url = URL.createObjectURL(blob);
Object.assign(document.createElement('a'), { href:url,
download:'data.json' }).click();
```

69) How do I read a stream line-by-line (SSE)?
Split by \n.

```
let buf=''; /* append chunk to buf; split on \n; process lines; keep
tail */
```

70) How do I convert Node Buffer ↔ base64?

```
const b64 = Buffer.from(buf).toString('base64');
const buf2 = Buffer.from(b64, 'base64');
```

I. Security & Safety (71–78)

71) How do I store API keys safely?
Never in client code; keep in server env vars.

```
// Node: process.env.OPENAI_API_KEY
```

72) How do I prevent prompt injection via user content?
Delimit input and instruct the model to treat it as untrusted.

```
const prompt = `SYSTEM: Follow policies.
USER_INPUT (verbatim, untrusted):
"""${userInput}"""`;
```

73) How do I sanitize HTML in model output?
Use a whitelist sanitizer (DOMPurify in browser) before `innerHTML`.

```
el.innerHTML = DOMPurify.sanitize(html);
```

74) How do I avoid XSS in code snippets?
Render as text, not HTML.

```
pre.textContent = codeString;
```

75) How do I implement role separation (system/dev/user)?
Pass explicit roles and isolate system prompts on server only.

76) How do I rate-limit per user/session?
Use IP/session + sliding window on server.

77) How do I encrypt at rest (Node)?

```
import crypto from 'crypto';
function enc(data, key){ const iv=crypto.randomBytes(12);
  const c=crypto.createCipheriv('aes-256-gcm', key, iv);
  const out=Buffer.concat([c.update(data,'utf8'), c.final()]);
  return { iv, out, tag:c.getAuthTag() };
}
```

78) How do I sign webhooks from AI providers?

Verify HMAC signature with shared secret before trusting payloads.

J. UI & Streaming UX (79–84)

79) How do I append streamed tokens to a chat window?

```
const out = document.querySelector('#out');
function append(text){ out.textContent += text; }
```

80) How do I keep the window scrolled to the latest token?

```
out.scrollTop = out.scrollHeight;
```

81) How do I render code blocks from model output safely?
Detect ``` fences and wrap in `<pre><code>` with `textContent`.

82) How do I show a typing indicator?

```
const dot = document.querySelector('#typing');
dot.hidden = false; /* hide when done */
```

83) How do I disable the Send button during a request?

```
btn.disabled = true; try { await send(); } finally { btn.disabled = false; }
```

84) How do I capture Enter to send and Shift+Enter for newline?

```
ta.addEventListener('keydown', e=>{
  if(e.key==='Enter' && !e.shiftKey){ e.preventDefault(); send(); }
});
```

K. Node.js & Servers (85–90)

85) How do I proxy AI calls from Node (hide key, handle CORS)?

```
import express from 'express';
const app = express();
app.use(express.json());
app.post('/ai', async (req,res)=>{
  const r = await fetch('https://api.example/model', {
    method:'POST',
    headers:{ 'content-type':'application/json', authorization:`Bearer ${process.env.KEY}` },
    body: JSON.stringify(req.body)
  });
  res.status(r.status).set('content-type', r.headers.get('content-type')||'application/json');
  r.body.pipe(res);
});
```

86) How do I enable CORS for your frontend?

```
import cors from 'cors';
app.use(cors({ origin: 'https://yourapp.com', credentials: true }));
```

87) How do I store chat history (SQLite quick start)?

```
import Database from 'better-sqlite3';
const db = new Database('data.db');
db.exec('create table if not exists msgs(id integer primary key, conv text, role text, content text)');
db.prepare('insert into msgs(conv,role,content) values (?,?,?)').run(convId,'user',content);
```

88) How do I paginate chat history?

```
const rows = db.prepare('select * from msgs where conv=? order by id
desc limit ? offset ?').all(convId, 50, 0);
```

89) How do I stream server-side to client (SSE)?

```
app.get('/stream', (req,res)=>{
  res.setHeader('Content-Type','text/event-stream');
  res.setHeader('Cache-Control','no-cache');
  const send = d => res.write(`data: ${d}\n\n`);
  send('hello'); /* flush tokens as they arrive */
});
```

90) How do I manage secrets locally (dotenv)?

```
import 'dotenv/config';
console.log(process.env.OPENAI_API_KEY);
```

L. Tools/Functions & Structured Output (91–95)

91) How do I ask the model to call a tool (function calling pattern)?
Return a schema; when the model requests a tool, validate and execute, then return results back.

```
// Pseudocode driver
if (modelReply.toolCall) {
  const args = validate(modelReply.toolCall.args);
  const result = await tools[modelReply.toolCall.name](args);
```

```
  return callModel({ messages: [...msgs, {role:'tool', name, content:
JSON.stringify(result)}]});
}
```

92) How do I strictly constrain the model to JSON?
Use system instruction + small JSON schema + parse/repair on failure.

93) How do I guard against "role leakage" in tool calls?
Never forward tool outputs as user/system; always `role:'tool'`.

94) How do I build a resilient "extract entities" tool?
Return arrays with stable ids and types, and validate shape before use.

95) How do I round-trip structured state (planner→executor)?
Keep a normalized state object; append deltas per turn; diff for UI.

M. Retrieval & RAG (96–100)

96) How do I split documents semantically (simple heuristic)?
Split by headings/blank lines, then by max chars.

```
function splitDoc(text, size=1500){
  return text.split(/\n\s*\n/).flatMap(p => p.length<=size? [p] :
chunk(p,size));
}
```

97) How do I store embeddings for search (in-memory demo)?

```
const store = []; // { id, vec, text }
store.push({ id:'1', vec: normalize(embed('hello')), text:'hello
world' });
```

98) How do I retrieve top-k and compose context?

```
const ctx = topK(queryVec, store,
```

```
5).map(x=>x.it.text).join('\n---\n');
```

99) How do I avoid stuffing too much context (token budget)?
Measure length and prune lowest-scoring chunks first.

100) How do I attribute sources in answers?
Attach `sourceId` with each chunk and include it in the final response footer.

Bonus: End-to-End Minimal Chat Flow (Browser + Node)

Client (send & stream):

```
async function sendMessage(prompt){
  const res = await fetch('/ai', { method:'POST',
headers:{'content-type':'application/json'}, body: JSON.stringify({
prompt }) });
  const reader = res.body.getReader(); const dec = new TextDecoder();
  while(true){
    const { value, done } = await reader.read();
    if(done) break;
    const chunk = dec.decode(value, { stream:true });
    append(chunk);
  }
}
```

Server (proxy to provider, stream back):

```
app.post('/ai', async (req,res)=>{
  const r = await fetch('https://api.example/generate', {
    method:'POST',
    headers:{
      'content-type':'application/json',
      authorization:`Bearer ${process.env.API_KEY}`
```

```
    },
    body: JSON.stringify({ model:'my-model', stream:true, input:
req.body.prompt })
  });
  res.status(200);
  r.body.pipe(res);
});
```