CERTLABZ CERTIFICATION BOARD

Certified JavaScript Application Developer (CJAD)

Ten Sample Questions

These ten questions are written at the difficulty of the live exam. Six carry a figure, a data table, a configuration or a code listing; four are plain scenario questions, as on the live paper. Every one is a scenario a working JavaScript application developer would meet, and none of them appears in the live question bank.

The live exam is multistage adaptive: how you score on one stage selects the difficulty of the next, so no two candidates sit the same paper. It is a performance-based exam: each form can carry up to 5 to 12 performance-based questions alongside the multiple-choice questions, which means you build, sequence or complete something rather than only choosing from a list. Answers and full explanations follow each question here; the live exam does not disclose item-level answers.

Question 1 — Domain 1: JavaScript Language Core — CONFIG

Emma Wilson runs the line below in the console.

```
console.log(typeof null);
```

Listing 1.1 A single evaluation.

What does this log?

A. object
B. null
C. undefined
D. boolean

Correct answer: A
In JavaScript typeof null returns the string object, a long-standing quirk of the language rather than a sign that null is an object. Code that checks for null must compare against null directly instead of relying on typeof.

Question 2 — Domain 2: Async and the Runtime — CONFIG

James Carter studies the order in which these lines log.

```
console.log(1);
setTimeout(() => console.log(2), 0);
Promise.resolve().then(() => console.log(3));
console.log(4);
```

Listing 2.1 Sync, microtask and macrotask.

In what order do the numbers appear?

A 1 2 3 4

B 1 4 3 2

C 1 3 4 2

D 4 3 2 1

Correct answer: B
The synchronous lines run first, logging 1 then 4. The microtask queue, which holds the resolved promise callback, drains before timers, so 3 logs next, and the setTimeout macrotask logs 2 last.

Question 3 Domain 3: DOM, Browser and Web APIs DATA

Olivia Davis compares what each DOM query call returns.

Call	Returns
querySelector	first match
querySelectorAll	static NodeList
getElementById	one element
getElementsByClassName	live collection

Table 3.1 DOM query calls and results.

Which call returns a static NodeList?

A querySelector

B getElementById

C querySelectorAll

D getElementsByClassName

Correct answer: C
querySelectorAll returns a static NodeList, a snapshot that does not update when the document changes. getElementsByClassName returns a live collection instead, while querySelector and getElementById each return a single element.

Question 4 Domain 4: Building Applications CONFIG

Liam Anderson reviews how one function is shared between files.

```
// util.js
export function sum(a, b) { return a + b; }
// app.js
import { sum } from "./util.js";
```

Listing 4.1 Sharing a function across modules.

How is sum exported here?

A. default export
B. global variable
C. CommonJS module
D. named export

Correct answer: D
Using export before the declaration and importing it inside braces makes this a named export. A default export would omit the braces on import, and neither a global variable nor a CommonJS module is in use here.

Question 5

Domain 5: Testing and Quality

DATA

Ava Thompson maps each kind of test to the scope it covers.

Test	Scope
Unit	one function
Integration	modules together
End to end	whole app
Regression	past bugs

Table 5.1 Test types and their scope.

Which test checks a single function in isolation?

A. End to end
B. Unit test
C. Integration
D. Regression

Correct answer: B
A unit test exercises one function or unit in isolation, which makes failures easy to locate. Integration tests cover modules working together, end to end tests drive the whole app, and regression tests guard against past bugs returning.

Question 6

Domain 6: Performance, Security and Deployment

CONFIG

David Brown finds this line, where userInput comes from a form.

```
el.innerHTML = userInput;
```

Listing 6.1 Writing user input into the page.

What is the risk, and the soundest fix?

A. XSS; use textContent
B. safe; no change needed
C. slow; cache the value
D. not an issue at all

Correct answer: A
Assigning untrusted input to innerHTML lets an attacker inject markup and script, a cross-site scripting risk. Setting textContent instead writes the value as plain text so any markup is shown, not executed.

Question 7

Domain 1: JavaScript Language Core

Comparing values with == can coerce types and hide bugs. What is the soundest habit for equality checks?

A. Always use ==
B. Use == for numbers
C. Prefer === for checks
D. Avoid all comparisons

Correct answer: C
The strict operator === compares value and type without coercion, so it avoids the surprising conversions that == performs. Preferring === for equality checks keeps comparisons predictable and is the widely recommended default.

Question 8

Domain 2: Async and the Runtime

An async function can reject, and the error is currently swallowed with no trace. What is the soundest practice?

A. Ignore all rejections
B. Block the main thread
C. Never use promises here
D. Await in try/catch

Correct answer: D
Awaiting the async call inside a try block lets the catch block handle a rejection, so the error is surfaced and dealt with rather than lost. Ignoring rejections hides failures, and blocking the thread or dropping promises is not a fix.

Question 9 Domain 3: DOM, Browser and Web APIs

A list grows to thousands of rows, each given its own click handler, and the page slows down. What is the soundest fix?

A. Add more handlers
B. Use event delegation
C. Remove all clicks
D. Poll the DOM each second

Correct answer: B
Event delegation attaches one handler to a shared parent and reads the target when a click bubbles up, so thousands of per-row handlers are replaced by a single listener. This lowers memory use and keeps handlers working as rows are added.

Question 10 Domain 6: Performance, Security and Deployment

A team hard-codes an API secret in front-end JavaScript that is shipped to every browser. What failed, and what is required?

A. Nothing is wrong here
B. Use a longer secret key
C. Keep secrets on server
D. Obfuscate the bundle

Correct answer: C
Anything shipped to the browser can be read by any user, so a secret in front-end code is effectively public. Secrets must stay on a server that the client calls, and obfuscation or a longer key does not make an exposed secret safe.

Published by the CertLabz Certification Board, a division of CertLabz LLC. Companion to objectives document CJAD-001. https://certlabz.com/certifications/certified-javascript-application-developer.html