100 Intermediate JavaScript Interview Questions: Promises, Async-Await & Call Stack

Promises (Questions 1-30)

Basic Promise Concepts

1. What is a Promise in JavaScript and what problem does it solve?
2. What are the three states of a Promise and explain each?
3. What is the difference between `Promise.resolve()` and `Promise.reject()`?
4. How do you create a Promise that resolves after a specific timeout?
5. What happens if you don't return anything from a `.then()` handler?

Promise Chaining

6. Explain how Promise chaining works and why it's useful.
7. What's the difference between returning a value vs returning a Promise in a `.then()` handler?
8. How do you handle errors in a Promise chain?
9. What happens if an error occurs in the middle of a Promise chain?
10. Can you chain `.catch()` handlers? What's the behavior?

Promise Methods

11. Explain the difference between `Promise.all()` and `Promise.allSettled()`.
12. When would you use `Promise.race()` vs `Promise.any()`?
13. What happens if one promise fails in `Promise.all()`?
14. How does `Promise.allSettled()` handle rejected promises?
15. What's the difference between `Promise.any()` and `Promise.race()`?

Advanced Promise Concepts

16. What is Promise microtask queue and how does it relate to the event loop?
17. How do you convert a callback-based function to return a Promise?
18. What is the difference between `.then()` and `.catch()` in terms of error handling?
19. Can you attach multiple `.then()` handlers to the same Promise? What happens?
20. What is Promise.resolve() useful for when the argument is already a Promise?

Promise Patterns

21. How do you implement a retry mechanism using Promises?

22. What is the "Promisification" pattern?
23. How do you implement Promise-based timeout functionality?
24. What's the difference between parallel and sequential Promise execution?
25. How do you handle multiple dependent API calls with Promises?

Promise Edge Cases

26. What happens when you throw an error inside a `.then()` handler?
27. Can a Promise be resolved multiple times? What happens if you try?
28. What's the behavior of `.finally()` in Promise chains?
29. How do unhandled Promise rejections work in different environments?
30. What happens if you return `undefined` from a `.then()` handler?

Async-Await (Questions 31-60)

Basic Async-Await

31. What is the relationship between async-await and Promises?
32. Can you use `await` without `async`? Why or why not?
33. What does an `async` function always return?
34. How do you handle errors with async-await?
35. What's the difference between `async function` and `function*` (generator)?

Error Handling

36. How do you handle multiple potential errors in async-await?
37. What happens if you don't handle an error in an async function?
38. Can you use try-catch with Promise chains? How?
39. How do you handle errors when using `await` with `Promise.all()`?
40. What's the difference between throwing an error and returning Promise.reject() in async functions?

Async-Await Patterns

41. How do you make multiple API calls in parallel using async-await?
42. What's the difference between sequential and parallel async operations?
43. How do you implement conditional async operations?
44. How do you convert a Promise chain to async-await?
45. What's the best way to handle optional async operations?

Advanced Async-Await

46. Can you use `await` inside a `.map()` callback? What are the implications?
47. How do you handle async operations in array methods like `forEach()`?
48. What is the top-level await and when can you use it?
49. How do async-await functions interact with the event loop?
50. Can you use `await` inside a `try` block and handle errors in `catch`?

Async-Await Performance

51. When should you prefer Promise chains over async-await?
52. How does async-await affect performance compared to Promises?
53. What are the memory implications of using async-await?
54. How do you optimize multiple independent async operations?
55. What's the impact of blocking vs non-blocking async operations?

Async-Await Edge Cases

56. What happens if you forget to use `await` with an async operation?
57. Can you use `await` inside a synchronous function?
58. How do you handle timeouts with async-await?
59. What happens when you `await` a non-Promise value?
60. How do you handle race conditions in async-await code?

Call Stack & Event Loop (Questions 61-85)

Call Stack Basics

61. What is the call stack and how does it work in JavaScript?
62. What happens when the call stack exceeds its limit?
63. How does JavaScript handle function calls internally?
64. What is the difference between call stack and execution context?
65. How do recursive functions affect the call stack?

Event Loop Fundamentals

66. Explain the JavaScript event loop and its components.
67. What is the difference between the call stack and the callback queue?
68. How does the event loop prioritize different types of tasks?
69. What are microtasks and how do they differ from macrotasks?

70. When does the event loop check the callback queue?

Task Queues

71. What is the difference between the callback queue and the microtask queue?
72. How do Promise callbacks get prioritized in the event loop?
73. What happens to `setTimeout` with 0 delay?
74. How do `setImmediate` and `process.nextTick` relate to the event loop?
75. What is the order of execution for different async operations?

Event Loop & Promises

76. Why do Promise callbacks execute before `setTimeout` callbacks?
77. How does the microtask queue affect Promise resolution?
78. What happens when you create multiple nested Promises?
79. How does `async-await` interact with the event loop?
80. What is the execution order when mixing Promises and timers?

Advanced Event Loop

81. How do browser environments and Node.js differ in event loop implementation?
82. What is event loop starvation and how can you prevent it?
83. How does the event loop handle long-running synchronous operations?
84. What happens to the call stack during async operations?
85. How do Web APIs interact with the JavaScript event loop?

Integration & Advanced Topics (Questions 86-100)

Combining Concepts

86. How do you debug Promise chains vs async-await code?
87. What are the performance implications of deeply nested Promises?
88. How do you handle memory leaks in Promise-heavy applications?
89. What's the best way to cancel ongoing async operations?
90. How do you implement backpressure in Promise-based systems?

Real-world Scenarios

91. How do you handle API rate limiting with Promises?
92. What's the best pattern for handling user authentication flows?
93. How do you implement progress tracking for multiple async operations?

94. What's the approach for handling real-time data updates?
95. How do you manage state in async-heavy React components?

Testing & Debugging

96. How do you unit test async functions effectively?
97. What tools help debug Promise chains and async-await code?
98. How do you mock async operations in tests?
99. What are common anti-patterns in Promise usage?
100. How do you profile and optimize async JavaScript performance?

Study Tips

For Promises:

- Practice creating Promises from scratch
- Understand the difference between resolved/fulfilled states
- Master error propagation in chains
- Learn all Promise static methods thoroughly

For Async-Await:

- Convert Promise chains to async-await and vice versa
- Practice error handling patterns
- Understand when to use parallel vs sequential execution
- Master debugging techniques

For Call Stack & Event Loop:

- Trace through code execution step by step
- Use browser dev tools to visualize the call stack
- Practice predicting execution order
- Understand browser vs Node.js differences

Practice Approach:

1. Start with basic concepts and gradually increase complexity
2. Write code examples for each question
3. Use browser dev tools to inspect execution
4. Practice explaining concepts out loud
5. Build small projects incorporating these concepts

Remember: Understanding the "why" behind each concept is more important than memorizing syntax!