JavaScript Team Evaluation

Advanced Level

Section A – Deep Conceptual Understanding

1. **Generators and Iterators:**
    - Explain in detail how yield and yield* differ in generator functions.
    - Provide an example where yield* delegation is necessary, and contrast what would happen if only yield was used.
    - What are the performance trade-offs between nested generators and flattening with yield*?
2. **The Spread and Rest Operators (...):**
    - The ... operator behaves differently in array literals, function parameters, and object literals. Demonstrate each case with working code and explain **why** the semantics differ.
    - What would happen if you try to spread a non-iterable into an array? How can this be handled gracefully?

Section B – Problem Solving & Applied Coding

4. **Async Generators:**
    - How would you cancel iteration early without leaking network requests?
5. **Regex Edge Case Debugging:**
    - Given the regex:

```
const regex = /^(\d+)*$/;
```

        - What is wrong with this pattern?
        - How could it lead to catastrophic backtracking?
        - Rewrite it to safely match only strings of digits.
6. **Object Manipulation:**
        - Why might deep copies with ... fail for nested structures? Propose safer alternatives.

Section C – Critical Thinking & System Design

7. **Memory Leaks in Generators:**
    - Consider a long-lived generator producing an infinite sequence (e.g., Fibonacci numbers).
    - Under what conditions could this design cause memory leaks in Node.js?
    - Suggest best practices for safely handling such unbounded streams.
8. **Regex in Production Systems:**
    - Suppose your team uses regex for input validation across multiple microservices.
    - What strategies would you adopt to ensure maintainability, readability, and security?
    - When would you **not** use regex, and what alternatives would you consider?
9. **The Spread Operator in Large Systems:**
    - Discuss potential pitfalls of overusing the spread operator in performance-critical applications.

Intermediary Level

Section A – Core Concepts (Short Theory)

1. What is the difference between var, let, and const in terms of scope and hoisting?
2. Explain the difference between == and ===. Why is === generally preferred?
3. What are closures in JavaScript? Give a real-world use case.
4. How does the JavaScript event loop handle asynchronous tasks like setTimeout?
5. What is the difference between synchronous and asynchronous execution?
6. Define higher-order functions. Why are they important in JavaScript?
7. What are arrow functions and how do they differ from normal functions in handling this?
8. Explain what promises are and how they improve async code readability.
9. What is the difference between null and undefined?
10. What is a pure function, and why is it useful in functional programming?

Section B – Identify & Explain (Code Snippets)

Q1:

```
console.log(a);

var a = 10;
```

```
console.log(b);

let b = 20;
```

- Explain why the first console.log works but the second throws an error.

Q2:

```
function test() {
  for (var i = 0; i < 3; i++) {
    setTimeout(() => console.log(i), 1000);
  }
}

test();
```

- What will be printed? Why?

Q3:

```
const obj = { name: "Sathvik" };

Object.freeze(obj);

obj.name = "Changed";

console.log(obj.name);
```

- What will be logged? Why?

Q4:

```
console.log([] == ![]);
```

- What will be the result? Explain with type coercion rules.

Q5:

```
function foo(a, b = 5) {
  return a + b;
}
```

```
console.log(foo(10));
```

- What will this print? What does the b = 5 mean?

Section C – Applied Understanding

1. What is event delegation and why is it useful in JavaScript applications?
2. Explain the difference between call, apply, and bind.
3. What is the difference between shallow copy and deep copy in JavaScript? Give an example of when it matters.
4. How does JavaScript handle prototype inheritance?
5. Explain the difference between map(), forEach(), and filter(). When would you use each?
6. What is a memory leak in JavaScript, and what practices help avoid it?
7. Explain the difference between function declaration and function expression.
8. How do import and require differ in JavaScript module systems?
9. What are JavaScript generators and when might you use them?
10. What is the difference between stack memory and heap memory in JavaScript execution?

Basic Level

Section A – Core Concepts (Basic Understanding)

1. What is JavaScript primarily used for in web development?
2. Explain the difference between var, let, and const.
3. What is the purpose of typeof in JavaScript?
4. What does == do differently than ===?
5. Name two primitive data types and one non-primitive data type in JavaScript.

Section B – Code Reading & Identification

6. Look at this:

```
let x = "5";

let y = 5;

console.log(x == y);
```

```
console.log(x === y);
```

- What will be printed and why?

7. What does the following return?

```
typeof null

typeof []
```

7. In the snippet below:

```
let fruits = ["apple", "banana", "cherry"];

console.log(fruits.length);
```

- What will console.log output?

9. What happens here?

```
let a;

console.log(a);
```

- What will be printed and why?

10. In the browser, what does alert("Hello!") do?

Section C – General Concepts

11. What is the difference between synchronous and asynchronous code in JavaScript?
12. What is the purpose of JSON.stringify() and JSON.parse()?
13. What is the DOM in JavaScript?
14. What is an event listener? Give one example event.
15. What is the use of localStorage in a web browser?

Miscellaneous Questions

1. What are the three states of a **Promise** in JavaScript?
2. How are **Promises** better than callbacks for handling asynchronous code?
3. What is the role of .then(), .catch(), and .finally() in a Promise?
4. Explain **async/await** in JavaScript. How does it simplify working with Promises?
5. How do you handle errors in **async/await** functions?
6. What is the difference between **synchronous** and **asynchronous** execution in JavaScript?

7. What is the **fetch API**? What does it return by default?
8. Explain the difference between **REST APIs** and normal JavaScript functions.
9. What is **event bubbling** and **event capturing** in JavaScript?
10. What is **event delegation** and why is it useful?
11. What is the difference between null, undefined, and NaN?
12. What is the effect of using "use strict" in JavaScript?
13. Differentiate between setTimeout() and setInterval().
14. What is **debouncing** in JavaScript? Give an example use case.
15. What is **throttling** in JavaScript? How is it different from debouncing?
16. What is the difference between a **Map** and an **Object** in JavaScript?
17. What is the difference between a **Set** and an **Array**?
18. Explain the use of **WeakMap** and **WeakSet** in JavaScript.
19. What is a **microtask** in the event loop? Give an example.
20. What is the purpose of the **finally block** in error handling?