.

🟦 Intermediate JavaScript Questions

1. What is the difference between undefined, null, and NaN?

- **undefined** means a variable has been declared but not assigned a value.
- **null** represents the intentional absence of any object value.
- **NaN** stands for “Not a Number” — it’s the result of an invalid mathematical operation.

---

2. What is hoisting?

JavaScript moves declarations (not initializations) to the top of their scope during compilation.
For example, variables declared with var are hoisted with undefined, while let and const are hoisted but remain in a *temporal dead zone* until initialized.

---

3. What is the difference between == and ===?

- **==** performs type coercion before comparison.
- **===** checks both value and type strictly.
  Always prefer === for cleaner, bug-free comparisons.

---

4. What is closure, and why is it useful?

A closure is formed when a function “remembers” the variables from its outer scope even after that scope has finished execution.
It helps in **data privacy**, **memoization**, and **function factories**.

---

5. Explain event bubbling and capturing.

In event bubbling, the event is handled first by the innermost target element, then propagates outward.
In event capturing (rarely used), it goes from the outermost element to the target element.
You can control this using the third parameter in addEventListener.

---

6. What is the difference between synchronous and asynchronous JavaScript?

- **Synchronous**: Executes line-by-line — blocks further execution until current task completes.
- **Asynchronous**: Non-blocking — uses callbacks, promises, or async/await to handle tasks concurrently.

---

7. How does JavaScript handle memory management?

JavaScript uses **automatic garbage collection**, cleaning up objects that are no longer referenced.
Developers should still avoid circular references and detached DOM nodes to prevent leaks.

---

8. What is the difference between shallow copy and deep copy?

- A **shallow copy** copies references to nested objects.
- A **deep copy** recursively copies all values, creating a fully independent clone.

---

9. What is the event loop?

The event loop allows JavaScript (single-threaded) to perform non-blocking I/O.
It continuously checks the **call stack** and **callback queue** to decide what to execute next.

---

10. Explain this keyword behavior.

this depends on **how a function is called**, not where it is defined.
In strict mode, unbound this is undefined; otherwise, it refers to the global object.

🟥 Advanced JavaScript Questions

1. What is prototype chaining?

All JavaScript objects inherit from a prototype.
When accessing a property, JS checks the object and then follows the **prototype chain** until it finds it or reaches Object.prototype.

---

2. What are generators and iterators?

Generators allow you to pause and resume function execution, producing values on demand.
Iterators define a standard way to access items sequentially, used by for...of and spread syntax.

---

3. Explain microtasks and macrotasks.

- **Microtasks**: Promise callbacks, queueMicrotask().
- **Macrotasks**: setTimeout, setInterval, I/O events.
  Microtasks always run before the next macrotask.

---

4. What are WeakMap and WeakSet used for?

They store object keys without preventing garbage collection — useful for caching or tracking without creating memory leaks.

---

5. What is a module pattern?

A design pattern that encapsulates private variables and exposes a public API.
Modern JS achieves this via **ES modules (import/export)**.

---

6. How does the JavaScript engine optimize performance?

JS engines like **V8** use Just-In-Time (JIT) compilation, hidden classes, and inline caching to speed up repeated code execution.

---

7. What is the difference between call stack, heap, and queue?

- **Call stack**: Tracks execution context.
- **Heap**: Stores objects and data.
- **Queue**: Holds asynchronous callbacks to be executed later.

---

8. What are proxies and how can they be used?

A Proxy allows you to intercept and redefine fundamental operations (get, set, etc.) on objects — useful for logging, validation, or virtual objects.

---

9. How does JavaScript handle concurrency?

Through the **event loop** and **non-blocking async APIs** — not real parallel threads, but task scheduling and callbacks managed by the runtime.

---

10. What are some common performance optimization techniques?

- Debouncing and throttling user input events
- Lazy loading resources
- Using web workers for heavy tasks
- Minimizing DOM reflows
- Caching results or API calls