1. Why were let and const introduced when var already existed?

Answer:
let and const were introduced to fix problems with var such as:

- Function scope instead of block scope
- Hoisting confusion
- Accidental redeclaration

let allows reassignment, const does not.

Syntax:

```
var a = 10;

let b = 20;

const c = 30;
```

---

2. How does JavaScript decide the data type of a variable at runtime?

Answer:
JavaScript is **dynamically typed**, meaning the data type is decided **at runtime** based on the value assigned to the variable.

Syntax:

```
let x = 10;      // number

x = "Hello";    // string
```

---

3. Difference between == and ===, and why companies prefer ===?

Answer:

- == compares **values only** (type conversion happens)
- === compares **value and type**

Companies prefer === because it avoids unexpected bugs.

Syntax:

```
5 == "5"   // true

5 === "5"  // false
```

---

4. How do logical operators (&&, ||) help in cleaner conditional code?

Answer:
Logical operators reduce the need for multiple if statements and make conditions short and readable.

Syntax:

```
if (age > 18 && hasID) {
  allowEntry();
}
```

---

5. When would you choose switch over if-else?

Answer:
switch is used when checking **one variable against many fixed values**, making the code more readable.

Syntax:

```
switch(day) {
  case "Mon":
    break;
  case "Tue":
    break;
  default:
    break;
}
```

---

6. Why does do-while execute at least once even if the condition is false?

Answer:
Because the condition is checked **after** the loop body runs.

Syntax:

```
do {
  console.log("Runs once");
} while (false);
```

---

7. What happens internally when a for loop runs?

Answer:

1. Initialization runs once
2. Condition is checked
3. Loop body executes
4. Increment/decrement runs
5. Steps repeat until condition is false

Syntax:

```
for (let i = 0; i < 3; i++) {
  console.log(i);
}
```

---

8. How does array indexing work, and what if index does not exist?

Answer:
Arrays use **zero-based indexing**.
If an index does not exist, JavaScript returns undefined.

Syntax:

```
let arr = [10, 20];
arr[0]; // 10
arr[5]; // undefined
```

---

9. Difference between map() and forEach()?

Answer:

Feature	**map()**	**forEach()**
Return value	New array	undefined
Use case	Transform data	Perform actions

Syntax:

```
arr.map(x => x * 2);

arr.forEach(x => console.log(x));
```

---

10. Why are higher-order array methods preferred over loops?

Answer:
They:

- Are easier to read
- Reduce code length
- Avoid manual index handling
- Improve maintainability

(No syntax required)

---

11. How is an object stored in memory compared to a primitive?

Answer:

- **Primitive values** are stored directly in stack memory
- **Objects** are stored in heap memory and accessed by reference

Example:

```
let a = 10;

let obj = { x: 1 };
```

---

12. Difference between for...in and Object.keys()?

Answer:

- for...in loops over enumerable properties (including inherited)
- Object.keys() returns only own property names

Syntax:

```
for (let key in obj) {

  console.log(key);
}

Object.keys(obj).forEach(key => console.log(key));
```

---

13. Difference between parameters and arguments, and callbacks?

Answer:

- **Parameters**: Variables in function definition
- **Arguments**: Actual values passed
- **Callback**: Function passed as an argument

Syntax:

```
function greet(name) {   // parameter
  name();
}

greet(() => console.log("Hello")); // argument (callback)
```

---

14. Why does asynchronous JavaScript not block the main thread?

Answer:
Async operations are handled by **Web APIs and the event loop**, allowing JavaScript to continue running and keeping the UI responsive.

(No syntax required)

---

15. Role of the V8 engine in JavaScript and React?

Answer:
V8:

- Compiles JavaScript to machine code
- Executes JS fast
- Supports React by efficiently handling rendering and execution