Modern JavaScript

ES6 Features Every Developer Must Know

Introduction: Why ES6 Matters

The release of ECMAScript 2015 (ES6) marked the most significant update to JavaScript since its inception. It transitioned the language from a simple scripting tool to a robust language capable of building complex, enterprise-level applications. For a developer, mastering ES6 is no longer optional; it is the industry standard. Interviewers use these topics to distinguish between someone who "knows" JavaScript and someone who understands how modern web development works.

---

1. let and const: Modern Block Scoping

Before ES6, var was the only way to declare variables. var is function-scoped and hoisted, often leading to bugs. ES6 introduced let and const.

Scope Differences

- **let**: Block-scoped. The variable is only available within the {} where it is defined.
- **const**: Block-scoped and immutable (the binding cannot change).
- **var**: Function-scoped. It ignores block boundaries like if or for.

Code Example

```
function scopeTest() {
  if (true) {
    var x = "var";
    let y = "let";
    const z = "const";
  }
  console.log(x); // "var"
  console.log(y); // ReferenceError: y is not defined
}
```

Common Mistakes

- **Assuming const makes objects immutable:** const prevents reassignment of the variable, but the *contents* of an object or array can still be modified.
- **Hoisting issues:** Unlike var, let and const are not initialized until their definition is reached (Temporal Dead Zone).

---

2. Arrow Functions: Concise Logic and Scope

Arrow functions provide a shorter syntax and handle the this keyword differently than traditional functions.

Syntax

```
// ES5
var multiply = function(a, b) {
  return a * b;
};

// ES6
const multiply = (a, b) => a * b;
```

'this' Behavior

In regular functions, this is determined by how the function is called. In arrow functions, this is **lexically bound**—it inherits this from the surrounding parent scope.

When Not to Use Arrow Functions

1. **Object Methods:** Since they don't have their own this, using them as methods will often result in this pointing to the global window.
2. **Event Listeners:** If you need this to refer to the element that triggered the event.

---

3. Template Literals: Beyond Quotes

Template literals use backticks (`) instead of quotes, allowing for easier string manipulation.

String Interpolation and Multiline Strings

```
const name = "John";
```

```
const role = "Developer";

// Old Way
const message = "Hello, " + name + ".\n" + "You are a " + role + ".";

// ES6 Way
const message = `Hello, ${name}.
You are a ${role}.`;
```

---

4. Destructuring: Extracting Data Efficiently

Destructuring allows you to "unpack" values from arrays or properties from objects into distinct variables.

Object Destructuring

```
const user = { id: 1, username: 'dev_guy', email: 'test@mail.com' };
// Extracting properties
const { username, email } = user;
```

Array Destructuring

```
const colors = ['red', 'green', 'blue'];
const [primary, secondary] = colors; // primary = 'red', secondary = 'green'
```

Real-World Example (React Props)

```
function Profile({ name, age }) {
  return `User: ${name}, Age: ${age}`;
}
```

---

5. Spread and Rest Operators

Both use the ... syntax but serve opposite purposes.

Spread Operator (Expanding)

Used to copy or merge arrays and objects.

```
const arr1 = [1, 2];
const arr2 = [...arr1, 3, 4]; // [1, 2, 3, 4]
const original = { a: 1 };
const copy = { ...original, b: 2 };
```

Rest Operator (Collecting)

Used in function parameters to collect multiple arguments into an array.

```
function sum(...numbers) {
  return numbers.reduce((acc, val) => acc + val, 0);
}
```

---

6. Default Parameters

You can now set default values for function parameters directly in the signature.

Usage

```
// If 'status' is not provided, it defaults to 'guest'
function login(user, status = 'guest') {
  console.log(`${user} logged in as ${status}`);
}
```

Interviewer Tip: Default parameters are only used if the argument is undefined. Passing null will not trigger the default value.

---

7. Enhanced Object Literals

ES6 simplifies object creation when property names match variable names.

Shorthand Properties and Methods

```
const model = "Tesla";
const year = 2023;
const car = {
  model, // Same as model: model
  year,
```

```
  drive() { // Shorthand for drive: function() {}
    console.log("Driving...");
  }
};
```

---

8. Modules (import / export)

ES6 Modules (ESM) allow you to break code into multiple files, improving maintainability.

Basic Usage

```
// math.js
export const add = (a, b) => a + b;
export default function multiply(a, b) { return a * b; }

// app.js
import multiply, { add } from './math.js';
```

Why Modules Matter

- **Encapsulation:** Variables are scoped to the module, not the global scope.
- **Tree Shaking:** Modern build tools can remove unused code to reduce bundle size.

---

9. ES6 Interview Questions & Common Traps

Q1: What is the Temporal Dead Zone (TDZ)?

Answer: It is the period between the entering of a scope and the actual declaration of a variable (let/const). Accessing the variable in this zone results in a ReferenceError.

Q2: Can you change a value inside an array declared with const?

Answer: Yes. const prevents the reassignment of the variable identifier to a new value/reference, but it does not make the object/array itself immutable.

Q3: How do arrow functions differ from regular functions regarding the arguments object?

Answer: Arrow functions do not have their own arguments object. They inherit it from the non-arrow parent function. To get all arguments in an arrow function, use the Rest operator (...args).

Q4: What is the difference between map() and forEach()?

Answer: map() returns a new array with transformed elements, while forEach() just executes a function for each element and returns undefined.

Quick Revision Table

Feature	Description	Key Benefit
let / const	Block-scoped declarations	Prevents variable leakage and hoisting bugs
Arrow Functions	Shorter syntax + Lexical this	Simplifies callbacks and keeps context
Destructuring	Unpacking objects/arrays	Cleaner, more readable code
Spread Operator	...obj to expand	Easy shallow copying and merging
Promises	Async handling	Replaces "Callback Hell"
Classes	Syntactic sugar for prototypes	Familiar OOP structure

---

Summary for Candidates

When answering ES6 questions in an interview:

1. **State the problem:** Explain why the old way (ES5) was problematic (e.g., "var caused scope leakage").
2. **State the solution:** Introduce the ES6 feature.
3. **Provide a scenario:** Give a real-world use case (e.g., "I use destructuring in React to clean up props").

Stay connected to get more useful resources like this :

Connect with The Boring Education:

- **Instagram**
- **LinkedIn**
- **Website**