2. ES6+ JavaScript for React

Last updated by | Subramanya Dixit | May 5, 2025 at 7:59 PM GMT+5:30

📘 Why ES6+ is Important for React

Modern React apps are built using the latest JavaScript (ES6+) syntax. Understanding these features is essential to writing clean, concise, and effective React code using Hooks and functional components.

🧠 Key Concepts

✅ let & const vs var

Use `let` and `const` for block-scoped variables. `const` is preferred for constants. Avoid `var` due to its confusing function scope.

✅ Arrow Functions

Provide shorter syntax and don't bind their own `this`, making them ideal for inline functions and React components.

```
const add = (a, b) => a + b;
```

✅ Destructuring

Simplifies access to object and array values, improving readability in props and state handling.

```
const person = { name: 'Alice', age: 20 };
const { name, age } = person;
```

✅ Spread & Rest Operators

Spread allows merging and copying data easily. Rest collects multiple values in functions.

```
const nums = [1, 2, 3];
const moreNums = [...nums, 4];
```

✅ Template Literals

Enable readable string construction with embedded expressions and multiline support.

```
const name = 'Alice';
console.log(`Hello, ${name}!`);
```

✅ Object & Array Enhancements

Simplify object definitions and operations with shorthand syntax and useful array methods.

```
const name = 'Bob';
const user = { name, age: 25 };
```

✅ Default + Named Imports/Exports

Facilitate modular development with flexible import/export structures.

```
// Export
export const add = (x, y) => x + y;
export default subtract;

// Import
import subtract, { add } from './math';
```

✅ Promises & Async/Await

Manage asynchronous operations clearly and concisely, avoiding nested callbacks.

```
const fetchData = async () => {
  const res = await fetch('/api/data');
  const json = await res.json();
  return json;
};
```

🧠 Visual Flowchart (Async/Await Flow)

```
flowchart TD
  A[Call Async Function] --> B[Await fetch()]
  B --> C[Wait for Promise to Resolve]
  C --> D[Process Response]
  D --> E[Return Data]
```

💡 Guidelines

- Prefer `const` by default; use `let` if reassignment is needed.
- Use arrow functions in components and callbacks.
- Destructure props and state for cleaner code.
- Handle async operations with `try/catch` and `await`.

📝 Practice Exercises

1. Use destructuring to extract values from an object and print them.
2. Write an arrow function that adds two numbers.
3. Combine two arrays using the spread operator.
4. Convert a regular function to use async/await syntax.

❓ Quiz Questions

1. What does `const` mean in ES6?
a) The variable can never be used again
✅ **b) The variable cannot be reassigned**
c) It's function-scoped
d) It's deprecated in modern JS

---

2. Which of the following is true about arrow functions?
a) They bind their own `this`
b) They are hoisted like regular functions
✅ **c) They do not bind their own `this`**
d) They can't be used inside components

---

3. What is destructuring in JavaScript?
a) Encrypting data
✅ **b) Extracting values from objects or arrays**
c) Rendering components
d) Binding functions to variables

---

4. Which syntax allows combining arrays?
a) `&` operator
✅ **b) Spread operator (`...`)**
c) `concat()` only
d) `+` operator
