📝 JavaScript ES6 & Advanced Concepts

---

1. Let, Const, and Var

Keyword	Scope	Reassignable	Hoisting
`var`	Function-scoped	Yes	Yes (undefined)
`let`	Block-scoped	Yes	No
`const`	Block-scoped	No	No

Example

```
function scopeExample() {
    if (true) {
        var x = 10; // function-scoped
        let y = 20; // block-scoped
        const z = 30; // block-scoped
    }
    console.log(x); // 10
    // console.log(y); // Error
    // console.log(z); // Error
}
scopeExample();
```

---

2. Arrow Functions

- Concise syntax, **lexical `this`** (does not bind `this`).

```
const add = (a, b) => a + b;
console.log(add(5, 3)); // 8
```

```
const greet = name => `Hello, ${name}!`;
console.log(greet("Aryan")); // Hello, Aryan!
```

- Lexical `this` example:

```
const person = {
    name: "Aryan",
    sayHi: function() {
        setTimeout(() => console.log(`Hi, I'm ${this.name}`), 1000);
    }
};
person.sayHi(); // Hi, I'm Aryan
```

---

3. Template Literals

- Use backticks ` to embed expressions.

```
const name = "Aryan";
const age = 22;
console.log(`My name is ${name} and I am ${age} years old.`);
```

- Multi-line strings:

```
const multiline = `Line 1
Line 2
Line 3`;
console.log(multiline);
```

---

4. Destructuring

Arrays

```
const arr = [1, 2, 3];
const [a, b, c] = arr;
console.log(a, b, c); // 1 2 3
```

Objects

```
const obj = { name: "Aryan", age: 22 };
const { name, age } = obj;
console.log(name, age); // Aryan 22
```

- Default values:

```
const { city = "Nagpur" } = obj;
console.log(city); // Nagpur
```

---

5. Spread & Rest Operators

- **Spread**: expands an array/object

```
const arr1 = [1,2];
const arr2 = [...arr1, 3,4];
console.log(arr2); // [1,2,3,4]

const obj1 = { a:1 };
const obj2 = { ...obj1, b:2 };
console.log(obj2); // { a:1, b:2 }
```

- **Rest**: collects remaining elements

```
const [first, ...rest] = [1,2,3,4];
console.log(first, rest); // 1 [2,3,4]

function sum(...nums) {
    return nums.reduce((a,b) => a+b, 0);
```

```
}
console.log(sum(1,2,3)); // 6
```

---

6. Default Parameters

```
function greet(name="Guest") {
    console.log(`Hello, ${name}`);
}
greet(); // Hello, Guest
greet("Aryan"); // Hello, Aryan
```

---

7. Enhanced Object Literals

- Shorthand properties and methods

```
const name = "Aryan";
const age = 22;
const person = { name, age, greet() { console.log("Hi!"); } };
console.log(person);
person.greet();
```

---

8. Classes & Inheritance

Class Example

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    greet() {
        console.log(`Hello, I'm ${this.name}`);
    }
```

```
}
const p = new Person("Aryan", 22);
p.greet(); // Hello, I'm Aryan
```

Inheritance

```
class Student extends Person {
    constructor(name, age, grade) {
        super(name, age);
        this.grade = grade;
    }
    study() {
        console.log(`${this.name} is studying`);
    }
}
const s = new Student("Aryan", 22, "A");
s.greet();
s.study();
```

---

9. Modules (ES6)

- **Export**

```
// math.js
export const add = (a,b) => a+b;
export const sub = (a,b) => a-b;
```

- **Import**

```
// app.js
import { add, sub } from './math.js';
console.log(add(2,3)); // 5
```

---

10. Promises & Async/Await Refresher (ES6+)

- Already covered in previous notes.
- Key methods: `.then()`, `.catch()`, `.finally()`, `Promise.all()`, `Promise.race()`.
- Async/await makes Promises **read like synchronous code**.

---

11. Map, Set, WeakMap, WeakSet

Map

```
let map = new Map();
map.set('name', 'Aryan');
console.log(map.get('name')); // Aryan
```

Set

```
let set = new Set([1,2,2,3]);
console.log(set); // {1,2,3}
```

WeakMap & WeakSet

- Key difference: keys are **objects**, garbage collected if no references.

---

12. Higher-Order Functions & Array Methods

```
const arr = [1,2,3,4];

// map
const doubled = arr.map(x => x*2); // [2,4,6,8]

// filter
```

```
const evens = arr.filter(x => x%2===0); // [2,4]

// reduce
const sum = arr.reduce((acc, val) => acc + val, 0); // 10
```

- Can pass functions as arguments and return functions → **functional programming style**.

---

13. Closures & Scope (Advanced)

Closure

```
function makeCounter() {
    let count = 0;
    return function() {
        count++;
        return count;
    };
}
let counter = makeCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

Lexical Scope & `this`

- Arrow functions **do not bind `this`**; regular functions do.

```
const obj = {
    name: "Aryan",
    greet: () => console.log(this.name), // undefined (arrow uses
outer scope)
    greet2() { console.log(this.name); } // Aryan
};
obj.greet();
obj.greet2();
```

---

14. Template for Fetch API (Advanced Use)

```
async function fetchData(url, options = {}) {
    try {
        const response = await fetch(url, options);
        if (!response.ok) throw new Error(`HTTP error! status:
${response.status}`);
        const data = await response.json();
        return data;
    } catch (error) {
        console.error("Fetch failed:", error);
    }
}

// Usage
fetchData('https://jsonplaceholder.typicode.com/todos/1')
    .then(data => console.log(data));
```

- Supports GET, POST, headers, JSON body.

---

15. Destructuring with Functions

```
function greet({name, age}) {
    console.log(`Hi, I'm ${name}, ${age} years old`);
}
greet({name: "Aryan", age: 22});
```

- Can also destructure arrays in function parameters.

---

16. Event Loop & Microtasks (Advanced JS Concept)

```
console.log("Start");

setTimeout(() => console.log("Timeout"), 0);

Promise.resolve().then(() => console.log("Promise"));

console.log("End");
```

Output

```
Start
End
Promise
Timeout
```

- **Explanation**: Promises go to the **microtask queue**, executed before **macrotasks** like `setTimeout`.

✅ Key Takeaways

- ES6 introduced **let/const, arrow functions, template literals, destructuring, spread/rest, classes, modules**.
- Advanced JS: **closures, higher-order functions, Map/Set, fetch API, event loop, microtasks/macrotasks**.
- Async JS + ES6 features together make **modern web development fluent**.
- **Practical usage:** All these are essential for **React, Node.js, and real-world web apps**.