JavaScript Map And ES-6

1.Introduction to ES6

ES6, also known as ECMAScript 2015, introduced several new features and syntax improvements to JavaScript. These changes make the language more powerful and easier to work with.

2. let and const

- **let**: Used to declare block-scoped variables.
- **const**: Used to declare block-scoped, read-only constants.

```
let name = 'Alice';
const age = 25;

name = 'Bob'; // This is allowed
age = 30; // This will cause an error
```

3. Arrow Functions

Arrow functions provide a shorter syntax for writing function expressions and lexically bind the `this` value.

```
// Traditional function
function add(a, b) {
  return a + b;
}

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

4. Template Literals

Template literals allow for easier string interpolation and multi-line strings using backticks.

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

const multiLine = `This is
a multi-line
string.`;
console.log(multiLine);
```

5. Default Parameters

Default parameters allow you to set default values for function parameters.

```
function greet(name = 'Guest') {
  console.log(`Hello, ${name}!`);
}

greet(); // Output: Hello, Guest!
greet('Alice'); // Output: Hello, Alice!
```

6. Destructuring Assignment

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

```
// Array destructuring
const numbers = [1, 2, 3];
const [a, b, c] = numbers;
console.log(a, b, c); // Output: 1 2 3

// Object destructuring
const person = { name: 'Alice', age: 25 };
```

```
const { name, age } = person;
console.log(name, age); // Output: Alice 25
```

7. Rest and Spread Operators

- **Rest**: Collects all remaining elements into an array.
- **Spread**: Spreads elements of an array or object.

```
// Rest operator
function sum(...numbers) {
  return numbers.reduce((acc, curr) => acc + curr, 0);
}
console.log(sum(1, 2, 3)); // Output: 6

// Spread operator
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5];
console.log(arr2); // Output: [1, 2, 3, 4, 5]

const obj1 = { a: 1, b: 2 };
const obj2 = { ...obj1, c: 3 };
console.log(obj2); // Output: { a: 1, b: 2, c: 3 }
```

8. Classes

ES6 introduced a class syntax for creating objects and dealing with inheritance.

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.
```

```
`);
  }
}

const person1 = new Person('Alice', 25);
person1.greet(); // Output: Hello, my name is Alice and I am 25 years old.
```

9. Promises

Promises provide a way to handle asynchronous operations more gracefully.

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('Data received');
  }, 2000);
});

promise.then((data) => {
  console.log(data); // Output: Data received
}).catch((error) => {
  console.error(error);
});
```

10. Modules

ES6 introduced a module system, allowing you to export and import functionalities between files.

export.js:

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

import.js:

```
import { add, subtract } from './export.js';
console.log(add(2, 3)); // Output: 5
```

```
console.log(subtract(5, 3)); // Output: 2
```

11. Enhanced Object Literals

Enhanced object literals provide a shorthand syntax for defining properties and methods in objects.

```
const name = 'Alice';
const age = 25;

const person = {
  name,
  age,
  greet() {
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.
`);
  }
};

person.greet(); // Output: Hello, my name is Alice and I am 25 years old.
```

12. Symbol

Symbols are a new primitive data type that can be used to create unique identifiers.

```
javascriptCopy code
const sym1 = Symbol('description');
const sym2 = Symbol('description');

console.log(sym1 === sym2); // Output: false
```

1. Introduction to Map

Map is a built-in object in JavaScript that allows you to store key-value pairs. Unlike objects, maps can use any value (including functions, objects, or any

primitive) as a key.

Key Features of Map:

- **Maintains insertion order**: Maps remember the original insertion order of the keys.
- **Key types**: Maps can use any type of values (both objects and primitive values) as keys.
- **Size property**: Maps have a `size` property that returns the number of elements.

2. Creating a Map

You can create a map using the `Map` constructor.

```
const map = new Map();
```

3. Adding Elements

You can add elements to a map using the `set` method.

```
map.set('name', 'Alice');
map.set('age', 30);
map.set('isStudent', true);
```

4. Accessing Elements

You can access elements in a map using the `get` method.

```
console.log(map.get('name')); // Output: Alice
console.log(map.get('age')); // Output: 30
console.log(map.get('isStudent')); // Output: true
```

5. Checking for Existence

You can check if a key exists in a map using the `has` method.

```
console.log(map.has('name')); // Output: true
console.log(map.has('address')); // Output: false
```

6. Removing Elements

You can remove elements from a map using the `delete` method.

```
map.delete('age');
console.log(map.has('age')); // Output: false
```

7. Clearing the Map

You can remove all elements from a map using the `clear` method.

```
map.clear();
console.log(map.size); // Output: 0
```

8. Iterating Over a Map

You can iterate over a map using various methods like `forEach`, `keys`, `values`, and `entries`.

Using `forEach`

```
map.set('name', 'Alice');
map.set('age', 30);

map.forEach((value, key) => {
  console.log(`${key}: ${value}`);
});
```