**ES6 Features**

**1. let & const**

**Description:**  
let and const are used to declare variables in a block-scoped manner, replacing the older var.

**Usage:**

* let is used when the variable value may change.
* const is used for variables that should not be reassigned.

**Syntax:**

let score = 50;

const username = "Shruthi";

**2. Arrow Functions**

**Description:**  
Arrow functions provide a concise syntax to write functions.

**Usage:**

* Makes code shorter and cleaner.
* Automatically binds this, which is useful in callbacks and React components.

**Syntax:**

const greet = (name) => `Hello, ${name}`;

**3. Template Literals**

**Description:**  
Template literals use backticks to create strings with embedded expressions.

**Usage:**

* Useful for dynamic strings and multiline content.
* Eliminates the need for string concatenation.

**Syntax:**

const name = "Shruthi";

console.log(`Welcome, ${name}!`);

**4. Destructuring**

**Description:**  
Destructuring allows extracting values from arrays or objects into variables.

**Usage:**

* Helps simplify code when accessing multiple values from a data structure.

**Syntax:**

const [a, b] = [10, 20];

const { name, age } = { name: "Shruthi", age: 22 };

**5. Spread and Rest Operators (...)**

**i) Spread Operator**

**Description:**  
The spread operator expands an array or object into individual elements. It’s mainly used for copying, merging, or passing values.

**Usage:**  
• Used to copy arrays or objects  
• Merge arrays or objects  
• Pass array elements as individual arguments to functions

**Syntax:**

// Copying an array

const arr1 = [1, 2, 3];

const arr2 = [...arr1]; // [1, 2, 3]

// Merging arrays

const a = [1, 2];

const b = [3, 4];

const combined = [...a, ...b]; // [1, 2, 3, 4]

// Copying and updating an object

const user = { name: "Shruthi" };

const updatedUser = { ...user, age: 22 }; // { name: 'Shruthi', age: 22 }

**ii) Rest Operator (...)**

**Description:**  
The rest operator **collects** multiple elements into a single array or object. It is mainly used in **function parameters** or **destructuring**.

**Usage:**  
• Used in functions to accept any number of arguments  
• Collects the “rest” of the values in array or object destructuring

**Syntax:**

**// Function arguments**

function sum(...numbers) {

return numbers.reduce((a, b) => a + b);

}

console.log(sum(1, 2, 3, 4)); // Output: 10

**// Array destructuring**

const [first, ...rest] = [10, 20, 30, 40];

console.log(first); // 10

console.log(rest); // [20, 30, 40]

**// Object destructuring**

const { name, ...others } = { name: "Shruthi", age: 22, city: "Bangalore" };

console.log(others); // { age: 22, city: "Bangalore" }

**6. Default Parameters**

**Description:**  
Functions can have default values for parameters. Functions takes the default values when no argument is passed.

**Usage:**

* Simplifies function calls and avoids undefined when no argument is passed.

**Syntax:**

function greet(name = "Guest") {

console.log(`Hello, ${name}`);

}

**7. for...of Loop**

**Description:**  
Used to iterate over iterable objects like arrays and strings.

**Usage:**

* Provides a cleaner way to loop through values directly.

**Syntax:**

const fruits = ["apple", "banana"];

for (let fruit of fruits) {

console.log(fruit);

}

**8. Promises**

**Description:**  
Promises are used to handle asynchronous operations like API requests.

**Usage:**

* Allows chaining .then() and .catch() for clean async logic.

**Syntax:**

fetch("https://api.example.com")

.then(res => res.json())

.then(data => console.log(data))

.catch(err => console.error(err));

**async and await**

**Description:**  
async/await is a cleaner way to work with Promises. It allows you to **write asynchronous code like it’s synchronous**, making it easier to read and understand.

**Usage:**

* async marks a function as asynchronous
* await pauses the code until the Promise is resolved
* Helps avoid chaining .then() and .catch()

**Syntax:**

async function getData() {

try {

const response = await fetch("https://api.example.com");

const data = await response.json();

console.log(data);

} catch (error) {

console.error(error);

}

}

getData();

**9. Classes**

A class is a blueprint for creating objects that share the same properties and methods. It helps organize your code and group related data and behavior together.

**Usage:**

* Used to create multiple objects with the same structure
* Makes code more structured and reusable, especially in large applications like e-commerce, games, or form management

**Syntax:**

class Person {

constructor(name) {

this.name = name;

}

greet() {

console.log(`Hi, I'm ${this.name}`);

}

}

const p1 = new Person("Shruthi");

p1.greet();

**10. Modules (import/export)**

**Description:**  
Modules allow splitting JavaScript code across multiple files.

**Usage:**

* Improves structure and reusability in large projects.

**Syntax:**

**greet.js**

export const greet = (name) => `Hi, ${name}`;

**main.js**

import { greet } from './greet.js';

console.log(greet("Shruthi"));

**11. Map in JavaScript**

**Description:**

A **Map** is a collection of key-value pairs where keys can be of any data type (not just strings like in objects). It preserves the **insertion order** of the keys.

**Usage:**

* Storing and accessing data with unique keys
* Useful when you need keys of any type (e.g., objects, functions)
* Better for frequently adding/removing key-value pairs compared to plain objects

**Syntax:**

const myMap = new Map();

myMap.set('name', 'Shruthi');

myMap.set('age', 22);

**Code Snippet:**

const userDetails = new Map();

userDetails.set('name', 'Shruthi');

userDetails.set('role', 'Developer');

console.log(userDetails.get('name')); // Output: Shruthi

console.log(userDetails.has('role')); // Output: true

userDetails.delete('role');

console.log(userDetails.size); // Output: 1

**Set in JavaScript**

**Description:**

A **Set** is a collection of **unique values** — it automatically removes duplicates. It also maintains the **insertion order**.

**Usage:**

* Storing a list of unique values
* Removing duplicates from arrays
* Useful in filtering, membership checks, or working with tags/categories

**Syntax:**

const mySet = new Set();

mySet.add(10);

mySet.add(20);

**Code Snippet:**

const numbers = new Set();

numbers.add(5);

numbers.add(10);

numbers.add(5); // Duplicate, won't be added again

console.log(numbers.has(10)); // Output: true

console.log(numbers.size); // Output: 2

numbers.delete(5);

console.log(numbers); // Output: Set { 10 }