ES6 JavaScript Features

# Features of ES6 (ECMAScript 2015)

* Let and Const Keywords: Provides block scope variable declaration.
* Arrow Functions: Shorter syntax for writing functions and lexically binds the `this` value.
* Classes: Introduces class syntax for object-oriented programming.
* Template Literals: Allows embedded expressions in strings using backticks.
* Destructuring Assignment: Enables unpacking values from arrays or properties from objects.
* Default Parameters: Allows setting default values for function parameters.
* Rest and Spread Operators: Useful for variable number of arguments and object/array cloning.
* Modules: `import` and `export` statements for modular code.
* Promises: Simplifies asynchronous code handling.
* Enhanced Object Literals: Shorthand syntax for object properties and methods.
* Map and Set: New data structures for efficient key-value and unique value storage.
* For...of Loop: Iterates over iterable objects like arrays, strings, etc.

# JavaScript let

The `let` keyword in JavaScript is used to declare variables that are limited to the scope of a block statement or expression. It was introduced in ES6 as an improvement over `var`.

Example:

let x = 10;  
if (x === 10) {  
 let x = 20;  
 console.log(x); // 20  
}  
console.log(x); // 10

Key Points:

* Block scoped.
* Cannot be redeclared in the same scope.
* Can be updated (reassigned).

# Differences between var and let

|  |  |  |
| --- | --- | --- |
| Feature | var | let |
| Scope | Function-scoped | Block-scoped |
| Hoisting | Yes, initialized as undefined | Yes, but not initialized |
| Redeclaration | Allowed | Not allowed in same scope |
| Temporal Dead Zone | No | Yes |

# JavaScript const

The `const` keyword declares variables that are block-scoped like `let`, but they cannot be reassigned after their initial assignment.

Example:

const x = 10;  
x = 20; // Error: Assignment to constant variable.

Key Points:

* Block scoped.
* Must be initialized during declaration.
* Cannot be reassigned, but if the value is an object or array, its contents can be modified.

# ES6 Class Fundamentals

ES6 introduced a new syntax for creating objects and dealing with inheritance using the `class` keyword. It is a syntactic sugar over JavaScript's existing prototype-based inheritance.

Example:

class Person {  
 constructor(name) {  
 this.name = name;  
 }  
 greet() {  
 console.log(`Hello, my name is ${this.name}`);  
 }  
}

# ES6 Class Inheritance

ES6 classes support inheritance using the `extends` keyword. The `super()` function is used to call the constructor of the parent class.

Example:

class Student extends Person {  
 constructor(name, grade) {  
 super(name);  
 this.grade = grade;  
 }  
 study() {  
 console.log(`${this.name} is studying in grade ${this.grade}`);  
 }  
}

# ES6 Arrow Functions

Arrow functions are a concise way to write function expressions in ES6. They also lexically bind the `this` context from the enclosing scope.

Example:

const add = (a, b) => a + b;  
console.log(add(2, 3)); // 5

Key Points:

* Shorter syntax compared to traditional functions.
* Do not have their own `this`, `arguments`, or `super`.
* Not suitable for use as constructors.

# set() and map()

\*\*Set:\*\*

A `Set` is a collection of unique values. Duplicate values are automatically removed.

Example:

const mySet = new Set([1, 2, 2, 3]);  
console.log(mySet); // Set { 1, 2, 3 }

\*\*Map:\*\*

A `Map` is a collection of key-value pairs where keys can be of any data type.

Example:

const myMap = new Map();  
myMap.set('name', 'John');  
myMap.set(1, 'one');  
console.log(myMap.get('name')); // John