**List the Features of ES6**

ES6 (ECMAScript 2015) introduced several powerful features that modernized JavaScript development:

**Core Language Features:**

* **let and const declarations** - Block-scoped variable declarations
* **Arrow functions** - Simplified function syntax with lexical this binding
* **Template literals** - String interpolation with backticks
* **Destructuring assignment** - Extract values from arrays and objects
* **Default parameters** - Set default values for function parameters
* **Rest and spread operators** - Handle variable numbers of arguments
* **Classes** - Object-oriented programming syntax
* **Modules** - Import/export functionality for code organization
* **Promises** - Better asynchronous programming
* **Enhanced object literals** - Shorthand property and method definitions

**Data Structures:**

* **Map and Set** - New collection types
* **WeakMap and WeakSet** - Garbage-collection friendly collections
* **Symbols** - New primitive data type

**Explain JavaScript let**

**let** is a block-scoped variable declaration introduced in ES6:

**Key Characteristics:**

* **Block scope**: Variables declared with let are only accessible within the block they're defined
* **No hoisting**: Cannot be accessed before declaration
* **No redeclaration**: Cannot declare the same variable twice in the same scope
* **Temporal dead zone**: Variables exist but are uninitialized until declaration

**Example:**

if (true) {  
 let x = 10;  
 console.log(x); // 10  
}  
console.log(x); // ReferenceError: x is not defined

**Identify the Differences Between var and let**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **var** | **let** |
| **Scope** | Function-scoped or globally-scoped | Block-scoped |
| **Hoisting** | Hoisted and initialized with undefined | Hoisted but not initialized |
| **Redeclaration** | Allows redeclaration in same scope | Does not allow redeclaration |
| **Temporal Dead Zone** | No | Yes |
| **Global Object Property** | Becomes property of global object | Does not become property of global object |

**Example:**

// var example  
function varExample() {  
 if (true) {  
 var x = 1;  
 }  
 console.log(x); // 1 - accessible outside block  
}  
  
// let example  
function letExample() {  
 if (true) {  
 let y = 1;  
 }  
 console.log(y); // ReferenceError - not accessible outside block  
}

**Explain JavaScript const**

**const** creates block-scoped constants that cannot be reassigned:

**Key Characteristics:**

* **Block-scoped**: Same scoping rules as let
* **Must be initialized**: Must assign value at declaration
* **No reassignment**: Cannot change the binding
* **Object mutability**: Objects/arrays can still be modified (contents can change)

**Examples:**

const PI = 3.14159;  
PI = 3.14; // TypeError: Assignment to constant variable  
  
const user = { name: 'John', age: 25 };  
user.age = 26; // This works - modifying object content  
user = {}; // TypeError: Assignment to constant variable

**Explain ES6 Class Fundamentals**

ES6 classes provide a cleaner syntax for creating objects and implementing inheritance:

**Basic Class Structure:**

class ClassName {  
 constructor(parameters) {  
 // Initialize properties  
 }  
   
 methodName() {  
 // Method implementation  
 }  
}

**Key Concepts:**

* **Constructor**: Special method for creating and initializing objects
* **Methods**: Functions defined inside the class
* **Properties**: Variables that belong to the class instance
* **Static methods**: Methods that belong to the class itself, not instances

**Example:**

class Person {  
 constructor(name, age) {  
 this.name = name;  
 this.age = age;  
 }  
   
 greet() {  
 return `Hello, I'm ${this.name}`;  
 }  
   
 static species() {  
 return 'Homo sapiens';  
 }  
}

**Explain ES6 Class Inheritance**

Class inheritance allows creating new classes based on existing classes using the extends keyword:

**Syntax:**

class ChildClass extends ParentClass {  
 constructor(parameters) {  
 super(parentParameters); // Call parent constructor  
 // Additional initialization  
 }  
   
 // Override or add new methods  
}

**Key Concepts:**

* **extends**: Keyword to create inheritance relationship
* **super()**: Calls the parent class constructor or methods
* **Method overriding**: Child class can redefine parent methods
* **Method inheritance**: Child inherits all parent methods

**Example:**

class Animal {  
 constructor(name) {  
 this.name = name;  
 }  
   
 speak() {  
 return `${this.name} makes a sound`;  
 }  
}  
  
class Dog extends Animal {  
 constructor(name, breed) {  
 super(name); // Call parent constructor  
 this.breed = breed;  
 }  
   
 speak() {  
 return `${this.name} barks`;  
 }  
}

**Define ES6 Arrow Functions**

Arrow functions provide a shorter syntax for writing functions with automatic this binding:

**Syntax Variations:**

// Traditional function  
function(a, b) { return a + b; }  
  
// Arrow function - multiple parameters  
(a, b) => a + b  
  
// Arrow function - single parameter  
x => x \* 2  
  
// Arrow function - no parameters  
() => console.log('Hello')  
  
// Arrow function - block body  
(a, b) => {  
 const sum = a + b;  
 return sum;  
}

**Key Features:**

* **Concise syntax**: Shorter than traditional functions
* **Implicit return**: Single expressions are automatically returned
* **Lexical this**: Inherits this from enclosing scope
* **No hoisting**: Cannot be called before definition

**Identify set() and map()**

**Set**

A **Set** is a collection of unique values:

**Key Features:**

* **Unique values only**: No duplicates allowed
* **Any data type**: Can store any type of values
* **Iterable**: Can be looped through

**Common Methods:**

* add(value) - Add a value
* delete(value) - Remove a value
* has(value) - Check if value exists
* clear() - Remove all values
* size - Get number of values

**Example:**

const mySet = new Set();  
mySet.add(1);  
mySet.add(2);  
mySet.add(2); // Duplicate - won't be added  
console.log(mySet.size); // 2

**Map**

A **Map** is a collection of key-value pairs:

**Key Features:**

* **Any data type keys**: Keys can be objects, primitives, functions
* **Maintains insertion order**: Iterates in insertion order
* **Size property**: Easy to get number of elements

**Common Methods:**

* set(key, value) - Add/update key-value pair
* get(key) - Retrieve value by key
* delete(key) - Remove key-value pair
* has(key) - Check if key exists
* clear() - Remove all entries
* size - Get number of key-value pairs

**Example:**

const myMap = new Map();  
myMap.set('name', 'John');  
myMap.set(1, 'number key');  
myMap.set(true, 'boolean key');  
console.log(myMap.get('name')); // 'John'  
console.log(myMap.size); // 3