### Exercise 9

### Features of ES6 (ECMAScript 2015)

ES6 introduced a wide range of new features to JavaScript to make the language more powerful and easier to use. Key features include:

* **let and const:** New keywords for declaring variables with block scope.
* **Arrow Functions:** A more concise syntax for writing function expressions.
* **Classes:** Syntactic sugar over JavaScript's existing prototype-based inheritance, making object-oriented programming more familiar to developers.
* **Modules:** The ability to import and export modules, allowing for better code organization and reuse.
* **Template Literals:** A new way to create strings that support multi-line strings and string interpolation using backticks (``).
* **Destructuring Assignment:** A syntax that makes it possible to unpack values from arrays or properties from objects into distinct variables.
* **Promises:** An object that represents the eventual completion or failure of an asynchronous operation and its resulting value.
* **for...of loop:** A new loop that iterates over iterable objects like arrays, strings, and sets.
* **Set and Map:** New data structures for storing unique values and key-value pairs, respectively.

### JavaScript let

The let keyword is used to declare a block-scoped local variable. This means the variable is only accessible within the block (defined by curly braces {}) where it is declared. Unlike var, which is function-scoped, let prevents variables from being accessed outside of their intended scope.

**Key Characteristics of let:**

* **Block Scope:** A variable declared with let is only available within the block in which it is defined.
* **No Redeclaration:** You cannot redeclare the same variable within the same scope using let. This helps prevent accidental overwrites.
* **Hoisting with a Temporal Dead Zone (TDZ):** While let declarations are hoisted to the top of their block, they are not initialized. Trying to access a let variable before its declaration will result in a ReferenceError.

### Differences Between var and let

|  |  |  |
| --- | --- | --- |
| Feature | var | let |
| **Scope** | Function-scoped. Accessible throughout the entire function. | Block-scoped. Accessible only within the block ({}) where it is declared. |
| **Hoisting** | Hoisted to the top of its function scope and initialized with undefined. | Hoisted to the top of its block scope but not initialized. Accessing it before declaration causes a ReferenceError (TDZ). |
| **Redeclaration** | Allows redeclaration of the same variable in the same scope, which can lead to bugs. | Does not allow redeclaration of the same variable in the same block, throwing a SyntaxError. |
| **Global Object** | When declared globally, var creates a property on the global object (window in browsers). | When declared globally, let does not create a property on the global object. |

### JavaScript const

The const keyword is used to declare a block-scoped constant, which means it must be initialized at the time of declaration and its value cannot be reassigned. This is crucial for creating immutable bindings.

**Key Characteristics of const:**

* **Block Scope:** Like let, const is block-scoped.
* **No Reassignment:** The value of a const variable cannot be changed after it is initialized.
* **Immutable Binding, Not Value:** It's important to understand that const creates an immutable binding. For primitive values (strings, numbers, booleans), this means the value itself is constant. However, for objects and arrays, the *reference* to the object/array is constant, but the properties or elements within the object/array can still be modified.

### ES6 Class Fundamentals

ES6 classes are a new syntax for creating objects and defining inheritance. They are essentially "syntactic sugar" over JavaScript's prototype-based inheritance model, providing a cleaner and more familiar way to structure code for those from class-based languages like Java or C++.

* **Defining a Class:** You define a class using the class keyword.
* **Constructor:** The constructor method is a special method for creating and initializing an object created with a class. It is executed when an instance of the class is created with the new keyword.
* **Methods:** Methods are defined inside the class and are shared by all instances of that class.

**Example:**

JavaScript

class Animal {  
  constructor(name) {  
    this.name = name;  
  }  
  
  speak() {  
    console.log(`${this.name} makes a noise.`);  
  }  
}  
  
const dog = new Animal('Dog');  
dog.speak(); // Output: Dog makes a noise.

### ES6 Class Inheritance

Class inheritance allows a new class (child class or subclass) to inherit properties and methods from an existing class (parent class or superclass).

* **extends keyword:** Used to create a child class from a parent class.
* **super() keyword:** In the child class's constructor, super() is used to call the parent class's constructor. This is mandatory if the child class has its own constructor.
* **Method Overriding:** A child class can redefine a method from the parent class to provide its own implementation.

**Example:**

JavaScript

class Dog extends Animal {  
  constructor(name, breed) {  
    super(name); // Call the parent constructor  
    this.breed = breed;  
  }  
  
  // Override the speak method  
  speak() {  
    console.log(`${this.name} barks.`);  
  }  
  
  getBreed() {  
    console.log(`The breed is ${this.breed}.`);  
  }  
}  
  
const myDog = new Dog('Max', 'Golden Retriever');  
myDog.speak(); // Output: Max barks.  
myDog.getBreed(); // Output: The breed is Golden Retriever.

### ES6 Arrow Functions

Arrow functions provide a more concise syntax for writing function expressions. They are defined using a "fat arrow" (=>).

**Key Characteristics:**

* **Concise Syntax:** They remove the need for the function keyword and, for single-line expressions, can omit the return keyword and curly braces.
* **No this binding:** Arrow functions do not have their own this context. Instead, they inherit the this value from the enclosing scope. This solves a common issue with traditional functions where this binding could be confusing.
* **Anonymous:** Arrow functions are inherently anonymous unless assigned to a variable.

**Example:**

JavaScript

// Traditional function expression  
const add = function(a, b) {  
  return a + b;  
};  
  
// Arrow function with concise syntax  
const addArrow = (a, b) => a + b;  
  
console.log(add(2, 3)); // 5  
console.log(addArrow(2, 3)); // 5

### Set() and Map()

Set and Map are two new data structures introduced in ES6.

* **Set():** A Set is a collection of unique values. It is useful for storing a list of elements where you only want each element to appear once.
* **Features:**
* Stores unique values only.
* You can add values using set.add(value).
* You can check for the existence of a value with set.has(value).
* The size property returns the number of elements.
* **Example:**  
  JavaScript  
  const uniqueNumbers = new Set([1, 2, 2, 3, 4, 4]);  
  console.log(uniqueNumbers); // Output: Set { 1, 2, 3, 4 }
* **Map():** A Map is a collection of key-value pairs, similar to an object, but with some key differences.
* **Features:**
* Allows keys of any data type (e.g., objects, functions, or any primitive value).
* Maintains the insertion order of its elements.
* You can add key-value pairs using map.set(key, value).
* You can retrieve a value using map.get(key).
* You can check for the existence of a key with map.has(key).
* The size property returns the number of elements.
* **Example:**  
  JavaScript  
  const myMap = new Map();  
  myMap.set('name', 'Alice');  
  myMap.set(1, 'one');  
    
  console.log(myMap.get('name')); // Output: Alice  
  console.log(myMap.get(1)); // Output: one

**IndianPlayers.js**

import React from 'react';

const OddPlayers = ({ players: [first, , third, , fifth] }) => {

  return (

    <div>

      <li>First: {first}</li>

      <li>Third: {third}</li>

      <li>Fifth: {fifth}</li>

    </div>

  );

};

const EvenPlayers = ({ players: [, second, , fourth, , sixth] }) => {

  return (

    <div>

      <li>Second: {second}</li>

      <li>Fourth: {fourth}</li>

      <li>Sixth: {sixth}</li>

    </div>

  );

};

const ListofIndianPlayers = ({ IndianPlayers }) => {

  return (

    <ul>

      {IndianPlayers.map((player, index) => (

        <li key={index}>{player}</li>

      ))}

    </ul>

  );

};

const IndianPlayersComponent = () => {

  const T20Players = ['First Player', 'Second Player', 'Third Player'];

  const RanjiTrophyPlayers = ['Fourth Player', 'Fifth Player', 'Sixth Player'];

  const IndianPlayers = [...T20Players, ...RanjiTrophyPlayers];

  return (

    <>

      <h1>Odd Players</h1>

      <OddPlayers players={IndianPlayers} />

      <hr />

      <h1>Even Players</h1>

      <EvenPlayers players={IndianPlayers} />

      <hr />

      <h1>List of Indian Players Merged:</h1>

      <ListofIndianPlayers IndianPlayers={IndianPlayers} />

    </>

  );

};

export { OddPlayers, EvenPlayers, ListofIndianPlayers };

export default IndianPlayersComponent;

**Scorebelow70.js**

import React from 'react';

const Scorebelow70 = ({ players }) => {

  const players70 = players.filter(item => item.score < 70);

  return (

    <ul>

      {players70.map((item, index) => (

        <div key={index}>

          <li>

            Mr. {item.name}<span>{item.score}</span>

          </li>

        </div>

      ))}

    </ul>

  );

};

export default Scorebelow70;

**ListofPlayers.js**

import React from 'react';

const ListofPlayers = ({ players }) => {

  return (

    <ul>

      {players.map((item, index) => (

        <div key={index}>

          <li>

            Mr. {item.name}<span>{item.score}</span>

          </li>

        </div>

      ))}

    </ul>

  );

};

export default ListofPlayers;

**App.js**

import React, { useState } from 'react';

import ListofPlayers from './components/ListofPlayers';

import Scorebelow70 from './components/Scorebelow70';

import { OddPlayers, EvenPlayers, ListofIndianPlayers } from './components/IndianPlayers';

import './App.css';

const App = () => {

  const [flag, setFlag] = useState(true);

  const players = [

    { name: 'Jack', score: 50 },

    { name: 'Michael', score: 70 },

    { name: 'John', score: 40 },

    { name: 'Ann', score: 61 },

    { name: 'Elisabeth', score: 61 },

    { name: 'Sachin', score: 95 },

    { name: 'Dhoni', score: 100 },

    { name: 'Virat', score: 84 },

    { name: 'Jadeja', score: 64 },

    { name: 'Raina', score: 75 },

    { name: 'Rohit', score: 80 },

  ];

  const T20Players = ['First Player', 'Second Player', 'Third Player'];

  const RanjiTrophyPlayers = ['Fourth Player', 'Fifth Player', 'Sixth Player'];

  const IndianTeam = [...T20Players, ...RanjiTrophyPlayers];

  return (

    <div className="App">

      <button onClick={() => setFlag(!flag)}>

        Toggle View (Current Flag: {flag ? 'true' : 'false'})

      </button>

      <hr />

      {flag ? (

        <div>

          <h1>List of Players</h1>

          <ListofPlayers players={players} />

          <hr />

          <h1>List of Players having Scores Less than 70</h1>

          <Scorebelow70 players={players} />

        </div>

      ) : (

        <div>

          <h1>Indian Team</h1>

          <h1>Odd Players</h1>

          <OddPlayers players={IndianTeam} />

          <hr />

          <h1>Even Players</h1>

          <EvenPlayers players={IndianTeam} />

          <hr />

          <h1>List of Indian Players Merged:</h1>

          <ListofIndianPlayers IndianPlayers={IndianTeam} />

        </div>

      )}

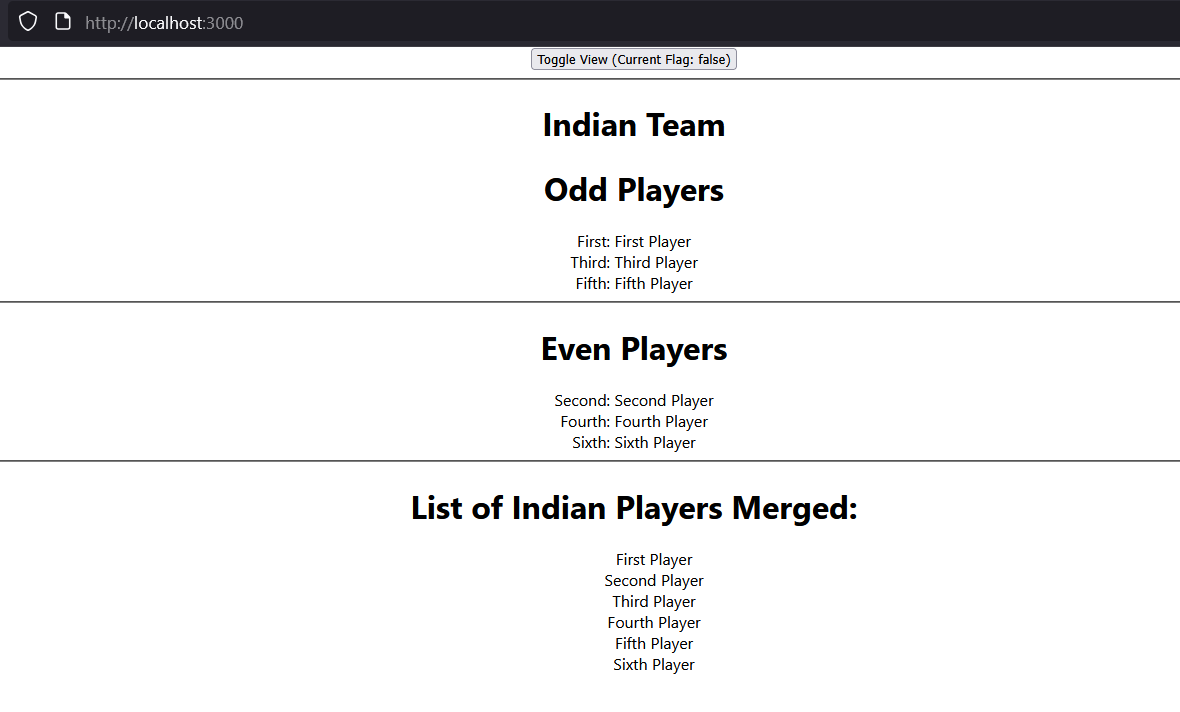
    </div>

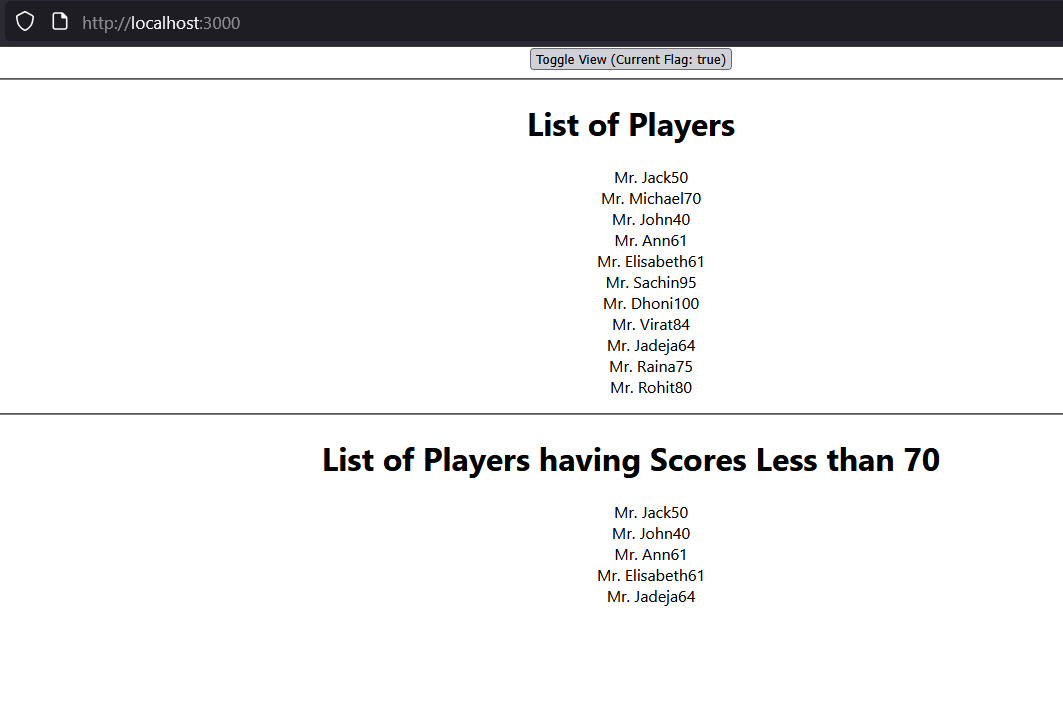
  );

};

export default App;

**Output:**

****

****