/\*

1. Explain callback function and higher order function with an example.

2. Explain map(), reduce(), filter() with an examples.

3. Explain Destructuring array and objects with example.

4.Expalin JSON ? Explain Why we use JSON.parse (), JSON.stringify() with an example.

5.Explain implicit and explicit conversions with an examples.

\*/

/\*Answer 1 starts \*/

/\*

Callback Function:

A callback function is a function that is passed as an argument to another function and is executed after the completion of some operation. Callbacks are commonly used in asynchronous programming, where functions may not complete their execution immediately, and you want to perform some action after the operation is finished.

\*/

// Function that takes a callback as an argument

*function* doSomethingAsync(*callback*) {

  setTimeout(*function* () {

    console.log("Operation completed!");

    callback(); // Call the callback function

  }, 2000);

}

// Callback function

*function* callbackFunction() {

  console.log("Callback executed!");

}

// Using the function with a callback

doSomethingAsync(callbackFunction);

/\*Higher-Order Function:

A higher-order function is a function that takes one or more functions as arguments, returns a function, or both. These functions enable a functional programming style in JavaScript and are often used to create more abstract and reusable code.

 \*/

// Higher-order function that takes a function as an argument

*function* multiplyBy(*factor*) {

  // Returns a new function that multiplies its argument by the given factor

  return *function* (*number*) {

    return *number* \* *factor*;

  };

}

// Usage of the higher-order function

*const* multiplyByTwo = multiplyBy(2);

*const* multiplyByFive = multiplyBy(5);

console.log(multiplyByTwo(4)); // Output: 8

console.log(multiplyByFive(3)); // Output: 15

/\*Answer 1 ends \*/

/\*Answer 2 starts \*/

/\*

1. map()

The map() function creates a new array by applying a provided function to each element in the original array.

 \*/

// Example using map() to double each element in an array

*const* numbers = [1, 2, 3, 4, 5];

*const* doubledNumbers = numbers.map(*function* (*num*) {

  return *num* \* 2;

});

console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]

/\*

2. reduce()

The reduce() function reduces an array to a single value by applying a function to each element and accumulating the results.

 \*/

// Example using reduce() to calculate the sum of elements in an array

*const* numbers1 = [1, 2, 3, 4, 5];

*const* sum = numbers1.reduce(*function* (*accumulator*, *currentValue*) {

  return *accumulator* + *currentValue*;

}, 0);

console.log(sum); // Output: 15

/\*

3. filter()

The filter() function creates a new array by applying a provided function to each element in the original array.

or

The filter() function creates a new array with elements that satisfy a provided condition.

 \*/

// Example using filter() to get even numbers from an array

*const* numbers3 = [1, 2, 3, 4, 5];

*const* evenNumbers = numbers3.filter(*function* (*num*) {

  return *num* % 2 === 0;

});

console.log(evenNumbers); // Output: [2, 4]

/\*Answer 2 ends \*/

/\*Answer 3 starts \*/

/\*

Destructuring Arrays:

Destructuring is a feature in JavaScript that allows you to extract values from arrays or properties from objects into distinct variables. It provides a concise and readable way to assign values.

Example 1:

 \*/

// Destructuring an array

*const* colors = ["red", "green", "blue"];

// Extracting values using destructuring

*const* [firstColor, secondColor, thirdColor] = colors;

console.log(firstColor); // Output: 'red'

console.log(secondColor); // Output: 'green'

console.log(thirdColor); // Output: 'blue'

/\* In this example, the values from the colors array are assigned to variables firstColor, secondColor, and thirdColor in the order they appear in the array. \*/

/\* Example 2 \*/

// Destructuring with default values

*const* fruits = ["apple"];

*const* [firstFruit, secondFruit = "orange"] = fruits;

console.log(firstFruit); // Output: 'apple'

console.log(secondFruit); // Output: 'orange' (default value used since the array has only one element)

/\*In this example, if there is no second element in the fruits array, the default value 'orange' is used. \*/

/\*

Destructuring Objects:

Example 1:

 \*/

// Destructuring an object

*const* person = {

  firstName: "Niraj",

  lastName: "Patil",

  age: 23,

};

// Extracting properties using destructuring

*const* { firstName, lastName, age } = person;

console.log(firstName); // Output: 'Niraj'

console.log(lastName); // Output: 'Patil'

console.log(age); // Output: 23

/\*Here, the properties of the person object are assigned to variables with the same names. \*/

/\* Example 2 \*/

// Destructuring with different variable names

*const* book = {

  title: "A Song of Ice and Fire",

  author: "George R. R. Martin ",

  year: 1996,

};

// Extracting properties with different variable names

*const* { title: bookTitle, author: bookAuthor, year: publicationYear } = book;

console.log(bookTitle); // Output: 'A Song of Ice and Fire'

console.log(bookAuthor); // Output: 'George R. R. Martin '

console.log(publicationYear); // Output: 1996

/\*

In this example, the properties of the book object are assigned to variables with different names using the colon (:) syntax.

Destructuring simplifies the process of extracting values from arrays or objects, making the code more concise and readable.

   \*/

/\*Answer 3 ends \*/

/\*Answer 4 Starts \*/

/\*

SON (JavaScript Object Notation):

JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is a text format that is completely language-independent but uses conventions familiar to programmers of the C family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.

JSON data is represented as key-value pairs, similar to object literals in JavaScript. The basic data types supported by JSON include objects, arrays, strings, numbers, booleans, and null.

JSON.parse() and JSON.stringify():

JSON.parse():

The JSON.parse() method is used to parse a JSON string and convert it into a JavaScript object.

\*/

// JSON string

*const* jsonString = '{"name": "Rohit", "age": 36, "city": "Mumbai"}';

// Parse JSON string to JavaScript object

*const* parsedObject = JSON.parse(jsonString);

console.log(parsedObject.name); // Output: 'Rohit'

console.log(parsedObject.age); // Output: 36

console.log(parsedObject.city); // Output: 'Mumbai'

/\* In this example, JSON.parse() is used to convert the JSON string into a JavaScript object (parsedObject). \*/

/\*

JSON.stringify():

The JSON.stringify() method is used to convert a JavaScript object into a JSON string.

 \*/

// JavaScript object

*const* person1 = {

  name: "MS Dhoni",

  age: 40,

  city: "Ranchi",

};

// Convert JavaScript object to JSON string

*const* jsonString1 = JSON.stringify(person1);

console.log(jsonString1);

// Output: '{"name":"MS Dhoni","age":40,"city":"Ranchi"}'

/\*In this example, JSON.stringify() is used to convert the person object into a JSON-formatted string (jsonString). \*/

/\*

Why Use JSON.parse() and JSON.stringify():

Data Exchange: JSON is commonly used for data exchange between a server and a web application. The server sends data in JSON format, and the client uses JSON.parse() to convert it into a JavaScript object for further manipulation.

LocalStorage and SessionStorage: When storing data in localStorage or sessionStorage, the data needs to be converted to a string. JSON.stringify() is used for this purpose. When retrieving the data, JSON.parse() is used to convert it back into a JavaScript object.

 \*/

// // Storing data in localStorage

// const dataToStore = { key: 'value' };

// localStorage.setItem('myData', JSON.stringify(dataToStore));

// // Retrieving data from localStorage

// const retrievedData = JSON.parse(localStorage.getItem('myData'));

/\*

API Communication: When working with APIs, data is often exchanged in JSON format. JSON.parse() is used to convert the received JSON response into a JavaScript object, and JSON.stringify() is used to convert a JavaScript object into a JSON string before sending it as part of the request.

These methods facilitate the interchange of data between different parts of a web application or between a web application and a server in a standardized and interoperable format.

Example:

 \*/

// <!DOCTYPE html>

// <html lang="en">

// <head>

//   <meta charset="UTF-8">

//   <meta name="viewport" content="width=device-width, initial-scale=1.0">

//   <title>Dark Mode Toggle</title>

//   <style>

//     body {

//       font-family: Arial, sans-serif;

//       padding: 20px;

//     }

//     /\* Dark mode styles \*/

//     body.dark-mode {

//       background-color: #333;

//       color: #fff;

//     }

//   </style>

// </head>

// <body>

//   <h1>Dark Mode Toggle</h1>

//   <label>

//     <input type="checkbox" id="darkModeToggle"> Dark Mode

//   </label>

//   <script>

//     // Retrieve user settings from localStorage

//     const storedSettings = JSON.parse(localStorage.getItem('userSettings')) || {};

//     // Apply dark mode setting if available

//     if (storedSettings.darkMode) {

//       document.body.classList.add('dark-mode');

//       document.getElementById('darkModeToggle').checked = true;

//     }

//     // Toggle dark mode when the checkbox is clicked

//     document.getElementById('darkModeToggle').addEventListener('change', function () {

//       // Update the user settings

//       const userSettings = {

//         darkMode: this.checked,

//       };

//       // Apply dark mode styles

//       if (userSettings.darkMode) {

//         document.body.classList.add('dark-mode');

//       } else {

//         document.body.classList.remove('dark-mode');

//       }

//       // Store the updated settings in localStorage

//       localStorage.setItem('userSettings', JSON.stringify(userSettings));

//     });

//   </script>

// </body>

// </html>

/\*Answer 4 ends \*/

/\*Answer 5 Starts \*/

// 1. Numeric to String Conversion:

// Implicit:

// Implicit conversion from number to string

*const* num = 42;

*const* strImplicit = "The answer is: " + num;

console.log(strImplicit); // Output: "The answer is: 42"

// Explicit:

// Explicit conversion from number to string using toString()

*const* num1 = 42;

*const* strExplicit = num1.toString();

console.log(strExplicit); // Output: "42"

// 2. String to Numeric Conversion:

// Implicit:

// Implicit conversion from string to number

*const* str = "123";

*const* numImplicit = str \* 1;

console.log(numImplicit); // Output: 123

// Explicit:

// Explicit conversion from string to number using parseInt()

*const* str1 = "456";

*const* numExplicit = parseInt(str1);

console.log(numExplicit); // Output: 456

// 3. Boolean to Numeric Conversion:

// Implicit:

// Implicit conversion from boolean to number

*const* bool = true;

*const* numImplicit1 = bool \* 1;

console.log(numImplicit1); // Output: 1

// Explicit:

// Explicit conversion from boolean to number using Number()

*const* bool1 = true;

*const* numExplicit1 = *Number*(bool1);

console.log(numExplicit1); // Output: 1

// 4. Numeric to Boolean Conversion:

// Implicit:

// Implicit conversion from number to boolean

*const* num2 = 42;

*const* boolImplicit = !!num2;

console.log(boolImplicit); // Output: true

// Explicit:

// Explicit conversion from number to boolean using Boolean()

*const* num3 = 42;

*const* boolExplicit = *Boolean*(num3);

console.log(boolExplicit); // Output: true

// 5. String to Boolean Conversion:

// Implicit:

// Implicit conversion from string to boolean

*const* str2 = "hello";

*const* boolImplicit1 = !!str2;

console.log(boolImplicit1); // Output: true (non-empty string is truthy)

// Explicit:

// Explicit conversion from string to boolean using Boolean()

*const* str3 = "hello";

*const* boolExplicit1 = *Boolean*(str3);

console.log(boolExplicit); // Output: true (non-empty string is truthy)

// 6. Object to Primitive Conversion:

// Objects can be implicitly or explicitly converted to primitive values using the valueOf() and toString() methods.

// Implicit:

// Implicit conversion from object to primitive (toString)

*const* obj = { key: "value" };

*const* strImplicit1 = "Object: " + obj;

console.log(strImplicit1); // Output: "Object: [object Object]"

// Explicit:

// Explicit conversion from object to primitive (toString)

*const* obj1 = { key: "value" };

*const* strExplicit1 = obj1.toString();

console.log(strExplicit1); // Output: "[object Object]"

/\*Answer 5 ends \*/