**Module:-1 Introduction of react js…**

**Q:-1 What is React.js? How is it different from other JavaScript frameworks and libraries?**

* React.js is a JavaScript library used to build user interfaces (UIs), especially for websites that update often, like Facebook or Instagram. It helps developers make reusable pieces of code called components.
* **How is React.js different from other libraries or frameworks?**
* **React is a library**, not a full framework like Angular.
* It uses something called the **Virtual DOM**, which makes updates on the web page faster.
* It uses **components**, which are small parts of the UI you can reuse.
* It lets data flow in **one direction**, which makes apps easier to manage.

**Q:-2: Explain the core principles of React such as the virtual DOM and component- based architecture.**

* **Virtual DOM**
* The DOM is the structure of your web page.
* The Virtual DOM is a lightweight copy of the real DOM that React uses in the background.
* When something changes (like clicking a button), React updates the Virtual DOM first, compares it to the old one, and then only changes what's needed in the real DOM.
* This makes your app faster and smoother.
* **Component-Based Architecture**
* React breaks the UI into **small, reusable pieces** called **components**.
* Each component controls **its own part** of the web page.
* You can **reuse** components anywhere in your app.
* **Example**:  
  A website page can be split into components like:
* Header
* Sidebar
* Button
* Footer

**function Welcome() {**

**return <h1>Hello, User!</h1>;**

**}**

**Q:-3: What are the advantages of using React.js in web development?**

* You can build small parts (like buttons or forms) and use them again in other places.
* React uses a Virtual DOM, so only the parts that change are updated.
* This makes the website load faster.
* React uses simple JavaScript and looks like HTML (using JSX).
* Beginners can learn it quickly.
* React is made by Facebook and has a large developer community.
* You can easily find help, tools, and tutorials.
* Data moves in one direction, making the app easy to control and debug.
* Many popular websites (like Facebook, Instagram, Netflix) use React.

**JavaScript Introduction**

**Q:-1 What is JavaScript? Explain the role of JavaScript in web development**.

**What is JavaScript?**

* **Definition:** A versatile scripting language that enables developers to add functionality to web pages.
* **Type:** Interpreted (runs without compiling) and dynamically typed.
* **Execution:** Runs in the browser using the JavaScript engine (e.g., Google Chrome’s V8 engine).
* **Standard:** Governed by the **ECMAScript** specification.

**Role of JavaScript in Web Development**

JavaScript plays three major roles in building modern websites:

**1. Adding Interactivity**

HTML gives structure, CSS styles it, and JavaScript brings it to life.  
Examples:

* Clicking a button to show/hide content
* Playing animations and transitions
* Form validations (checking email format before submission)

<button onclick="alert('Hello!')">Click Me</button>

**2. Manipulating the DOM (Document Object Model)**

JavaScript can dynamically update, add, or remove HTML elements without reloading the page.

**Example:**

document.getElementById("title").innerText = "Welcome to JavaScript!";

**3. Enabling Dynamic Content & Logic**

JavaScript can handle logic, calculations, and API calls to fetch and display live data.  
Examples:

* Displaying real-time weather updates
* Loading new products without refreshing
* Building single-page applications (SPAs) with frameworks like React, Angular, or Vue.

**Q:-2 How is JavaScript different from other programming languages like Python or Java?**

JavaScript differs from other programming languages like Python or Java in several key aspects:

* **Primary Domain and Execution Environment:**
  + **JavaScript:**Primarily designed for web browsers to create interactive and dynamic client-side web content. It's interpreted directly by the browser. With Node.js, it can also be used for server-side development.
  + **Python:**A general-purpose language used for a wide range of applications, including web development (backend), data science, machine learning, automation, and scripting. It's typically interpreted.
  + **Java**: A high-level, class-based, object-oriented language primarily used for enterprise-level applications, Android mobile development, and large-scale systems. It iscompiled into bytecode and then executed by the Java Virtual Machine (JVM).
* **Typing:**
  + **JavaScript:**Dynamically and weakly typed. Variable types are determined at runtime, and implicit type conversions are common.
  + **Python:**Dynamically but strongly typed. Variable types are determined at runtime, but implicit type conversions are less common and typically require explicit casting. Python 3.5 introduced optional type annotations for improved readability and static analysis.
  + **Java:**Statically and strongly typed. Variable types must be explicitly declared at compilation time, and type checking is performed during compilation.
* **Compilation vs. Interpretation:**
  + **JavaScript:**Primarily an interpreted language, though modern JavaScript engines use Just-In-Time (JIT) compilation for performance optimization.
  + **Python:**An interpreted language.
  + **Java:**A compiled language. Source code is compiled into bytecode, which is then interpreted by the JVM.
* **Concurrency Model:**
  + **JavaScript:**Historically single-threaded with an event loop for asynchronous operations, allowing non-blocking I/O. Web Workers provide limited multi-threading capabilities.
  + **Python:**Single-threaded by default due to the Global Interpreter Lock (GIL), which limits true parallel execution of threads for CPU-bound tasks, though itsupports concurrency through threading and asynchronous programming.
  + **Java:**Designed for multi-threading and provides robust mechanisms for concurrent programming.
* **Syntax and Paradigms:**
  + **JavaScript:**C-style syntax, supports multiple paradigms including object-oriented (prototype-based), functional, and imperative**.**
  + **Python:**Emphasizes readability with significant whitespace for code blocks. Supports object-oriented, imperative, and functional programming.
  + **Java:**C-style syntax, primarily object-oriented (class-based).

**Q:-3 Discuss the use of <script> tag in HTML. How can you link an external JavaScript file to an HTML document?**

The <script> tag in HTML is used to embed or reference executable code, primarily JavaScript, within an HTML document. It allows for dynamic and interactive content on web pages.

Key uses of the <script> tag:

* **Embedding inline JavaScript:** JavaScript code can be directly written within the opening and closing <script> tags.

<script>  
 document.getElementById("myElement").innerHTML = "Hello, World!";  
 </script>

* **Linking external JavaScript files:** The src attribute is used to specify the path to an external JavaScript file, promoting code organization and reusability.

Linking an external JavaScript file to an HTML document:

To link an external JavaScript file, the <script> tag is used with the src attribute, whose value is the path to the JavaScript file.

**<!DOCTYPE html>  
<html>  
<head>  
 <title>External JavaScript Example</title>  
</head>  
<body>  
 <h1>My Web Page</h1>  
 <p id="message"></p>  
  
 *<!-- Linking an external JavaScript file -->*  
 <script src="path/to/your/script.js"></script>   
</body>  
</html>**

**Placement of the <script> tag:**

* **In the <head> section:**

Scripts placed here are downloaded and executed before the page content is rendered. This can block rendering if the script is large or slow, but is suitable for scripts that need to be available early, such as those manipulating the DOM before content loads (with defer or async attributes).

* **Before the closing </body> tag:**

This is a common practice as it allows the HTML content to load and render first, improving perceived page load times. The JavaScript then executes after the HTML is available.

Attributes for optimization:

* **async:**

This boolean attribute allows the script to be downloaded asynchronously while the HTML parsing continues. The script executes as soon as it's downloaded, potentially out of order.

* **defer:**

This boolean attribute also allows asynchronous download, but ensures the script executes only after the HTML document has been fully parsed and rendered, and in the order they appear in the HTML. This is often preferred for scripts that interact with the DOM.

**Variables and Data Types**

**Q:-1What are variables in JavaScript? How do you declare a variable using var, let, and const?**

**Variables in JavaScript**

* **Definition:** A variable is a named container used to store data values in a program.
* In JavaScript, variables can hold different data types like numbers, strings, booleans, objects, etc.
* Variables help in storing, updating, and reusing values in the code.

**Declaring Variables**

JavaScript provides three keywords to declare variables: var, let, and const.

**1. var**

* Old way of declaring variables (before ES6).
* Function-scoped or globally scoped (NOT block-scoped).
* Can be re-declared and updated**.**

**var name = "John";**

**var name = "Alice";**

**name = "Bob";**

**2. let**

* Introduced in ES6 (2015).
* Block-scoped (limited to {} where it’s defined).
* Can be updated but not re-declared in the same scope.

**let age = 25;**

**age = 30;**

**3. const**

* Introduced in ES6.
* Block-scoped.
* Cannot be updated or re-declared (value stays constant).
* Must be initialized at the time of declaration.

**const country = "India";**

**Q:-3 Explain the different data types in JavaScript. Provide examples for each.**

JavaScript, a dynamically typed language, automatically determines the data type of a variable at runtime based on the assigned value. Data types can be classified into two main categories: primitive and non-primitive (or reference).

**1. Primitive data types**

**These data types represent single values and are immutable.**

* **Number:**Represents numeric values, including integers, decimals, and floating-point numbers. It also includes special values like Infinity, -Infinity (when a positive or negative number is divided by zero), and NaN (Not-a-Number).

**let integerNumber = 42;**

**let floatNumber = 3.14;**

**let result = 1 / 0;**

**let invalidOperation = "hello" \* 2;**

* **String:**Represents textual data (sequences of characters) enclosed in single quotes (' '), double quotes (" "), or backticks ( ).

**let greeting = "Hello, World!";**

**let name = 'Alice';**

**let message = `My name is ${name}`;**

* **Boolean:**Represents a logical entity with two possible values: true or false.

**let isActive = true;**

**let hasPermission = false;**

**Undefined:**A variable that has been declared but not assigned a value has the value and type undefined.

**let myVariable;**

**Q:2 What is the difference between undefined and null in JavaScript?**

In JavaScript, both undefined and null represent the absence of a value, but they differ in their origin and intended use.

**undefined:**

* **Meaning:**undefined indicates that a variable has been declared but has not yet been assigned a value. It can also represent the absence of a property in an object or a function parameter that was not provided.
* **Origin**: undefined is automatically assigned by the JavaScript engine in various scenarios where a value is expected but not explicitly provided.
* **Type:**typeof undefined returns "undefined".

**null:**

* **Meaning:** null represents an intentional absence of any object value. It signifies that a variable or property explicitly holds "no value" or "nothing."
* **Origin:** null must be explicitly assigned by the developer. It is not automatically assigned by the JavaScript engine.
* **Type:** typeof null returns "object". This is a long-standing quirk in JavaScript and is not indicative of null being an object in the traditional sense.

**JavaScript Operators**

**Q:1 What are the different types of operators in JavaScript? Explain with examples.**

**Arithmetic operators**

**Assignment operators**

**Comparison operators**

**Logical operators**

**1. Arithmetic Operators**

* Used to perform mathematical calculations.
* They work on numeric values and return a numeric result.
* Examples include:
  + + Addition
  + - Subtraction
  + \* Multiplication
  + / Division
  + % Modulus (remainder)
  + \*\* Exponentiation
  + ++ Increment
  + -- Decrement
* **Example**

**let x = 10, y = 4;**

**console.log(x + y); // 14**

**console.log(x % y); // 2**

**2. Assignment Operators**

* Used to assign values to variables.
* Can also perform an operation and assign the result in a single step.
* **Examples include:**
  + = Assign value
  + += Add and assign
  + -= Subtract and assign
  + \*= Multiply and assign
  + /= Divide and assign
  + %= Modulus and assign
  + \*\*= Exponentiation and assign
* **Example**

**let num = 5;**

**num += 2; // 7**

**num \*= 3; // 21**

**3. Comparison Operators**

* Used to compare two values and return a Boolean result (true or false).
* They are often used in decision-making (if conditions).
* **Examples include:**
  + == Equal to (compares values, ignores type)
  + === Strict equal (compares values and type)
  + != Not equal to (ignores type)
  + !== Strict not equal (compares value and type)
  + > Greater than
  + < Less than
  + >= Greater than or equal to
  + <= Less than or equal to
* **Example**

**console.log(10 > 5); // true**

**console.log(5 === "5"); // false**

**4. Logical Operators**

* Used to combine or invert conditions.
* Commonly used in conditional statements to make complex decisions.
* **Examples include:**
  + && Logical AND (true only if both conditions are true)
  + || Logical OR (true if at least one condition is true)
  + ! Logical NOT (reverses the Boolean value)
* **Example**

**let age = 20;**

**console.log(age > 18 && age < 30); // true**

**console.log(age < 18 || age > 25); // false**

**console.log(!(age > 25)); // true**

**Q:-2 What is the difference between == and === in JavaScript?**

| **==** | **===** |
| --- | --- |
| Performs type coercion (converts values to a common type before comparison). | No type coercion. Values must be of the same type and value to be equal. |
| Compares only values after converting them to the same type. | Compares both value and type exactly. |
| Used when you want JavaScript to automatically convert types for you. | Used for strict comparisons when you want to ensure that both type and value are identical. |
| '5' == 5 returns true (because it converts '5' to 5). | 5' === 5 returns false (different types: string vs number). |
| true == 1 returns true (because true is converted to 1). | true === 1 returns false (different types: boolean vs number). |
| null == undefined returns true. | null === undefined returns false. |
| Objects or arrays are compared by reference (not content). | Objects or arrays are compared by reference (not content). |

**Control Flow (If-Else, Switch)**

**Q:1 What is control flow in JavaScript? Explain how if-else statements work with an example.**

Control flow in JavaScript dictates the order in which statements are executed within a program. By default, code runs sequentially from top to bottom. However, control flow statements, such as conditionals and loops, allow for deviation from this linear execution based on specific conditions or for repetitive actions.

The if-else statement is a fundamental conditional control flow structure that allows a program to make decisions. It executes one block of code if a specified condition evaluates to true, and an alternative block of code if the condition evaluates to false.

**How if-else works:**

* **if statement:** The program first evaluates the condition provided within the parentheses of the if statement.
* **True condition:** If the condition is true, the code block immediately following the if statement (enclosed in curly braces {}) is executed.
* **False condition:** If the condition is false, the code block within the if statement is skipped, and the program proceeds to the else statement (if present).
* **else statement:** The else statement provides an alternative code block to be executed when the if condition is false.

**Example:**

**let temperature = 25;  
  
if (temperature > 30) {  
 console.log("It's a hot day!");  
} else {  
 console.log("The weather is moderate.");  
}**

**Q:-2 Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?**

**Switch Statement in JavaScript**

* The switch statement is used to execute one block of code from multiple possible options.
* It compares the value of an expression against multiple case labels.
* When a match is found, the code inside that case is executed until a break statement is encountered.
* If no case matches, the default block (if present) is executed.

**How It Works (Theory Steps)**

1. The expression inside switch() is evaluated once.
2. The result is compared strictly (===) with each case value.
3. If a match is found, that case’s code runs until a break or end of the statement.
4. If no match is found, the default block runs (if provided).

**When to Use switch Instead of if-else**

**Use a switch statement when:**

* You need to compare one single value against many possible fixed values.
* You want cleaner and more readable code instead of writing multiple else if conditions.
* Example: Menus, day-of-week checks, status codes, etc.

**Use if-else when:**

* You need to test complex conditions (ranges, multiple variables, logical expressions).
* The conditions are not just equality checks.

**Loops (For, While, Do-While)**

**Q:-1 Explain the different types of loops in JavaScript (for, while, do-while). Provide abasic example of each.**

**Loops in JavaScript**

* **Definition:** Loops are used to execute a block of code repeatedly as long as a condition is true.
* They help avoid writing repetitive code.
* Common types: for, while, do-while**.**

**1. for Loop**

* Used when the number of iterations is known in advance.
* Has three parts: initialization, condition, increment/decrement.

**Syntax:**

**for (initialization; condition; update) {**

**// code to run**

**}**

**Example:**

**for (let i = 1; i <= 5; i++) {**

**console.log(i);**

**}**

**// Output: 1 2 3 4 5**

**2. while Loop**

* Used when the number of iterations is unknown and depends on a condition.
* Checks the condition before each iteration.

**Syntax:**

**while (condition) {**

**// code to run**

**}**

**Example:**

**let i = 1;**

**while (i <= 5) {**

**console.log(i);**

**i++;**

**}**

**// Output: 1 2 3 4 5**

**3. do-while Loop**

* Similar to while, but executes the code at least once before checking the condition.
* Condition is checked after each iteration.

**Syntax:**

**do {**

**// code to run**

**} while (condition);**

**Example:**

**let i = 1;**

**do {**

**console.log(i);**

**i++;**

**} while (i <= 5);**

**// Output: 1 2 3 4 5**

**Q:-2What is the difference between a while loop and a do-while loop?**

* **while loop :**
  + Condition Check: The condition is checked *before* the loop's body is executed in each iteration.
  + Execution Guarantee: If the condition is initially false, the loop's body will not execute even once.
  + Example: A while loop is suitable for situations where you might not need to execute the loop at all, depending on an initial condition, [like checking if a file exists before processing](https://www.shiksha.com/online-courses/articles/difference-between-while-and-do-while-loop/).
  + Syntax (C-style): while (condition) { // loop body }
* **do-while loop :**
  + Condition Check: The loop's body is executed *at least once*, and then the condition is checked *after* the body's execution.
  + Execution Guarantee: The loop's body is guaranteed to execute at least once, even if the condition is initially false.
  + Example: A do-while loop is useful when you need to perform an action at least once, like [getting user input](https://www.shiksha.com/online-courses/articles/difference-between-while-and-do-while-loop/) and then validating it.
  + Syntax (C-style): do { // loop body } while (condition);

**Functions**

**Q:-1 What are functions in JavaScript? Explain the syntax for declaring and calling a function.**

**Functions in JavaScript**

**Definition**

* A function is a reusable block of code that performs a specific task.
* It allows you to avoid repeating code and makes programs easier to read, maintain, and debug.
* A function can take inputs (parameters) and return an output (return value).

**Syntax for Declaring a Function**

**function function Name(parameters) {**

**// code to be executed**

**return value; // optional**

**}**

**Syntax for Calling a Function**

**Function Name(arguments);**

* arguments are the actual values passed to the function parameters.

**Example**

**// Function Declaration**

**function greet(name) {**

**console.log("Hello, " + name + "!");**

**}**

**// Function Call**

**greet("John");**

**Q:-2 What is the difference between a function declaration and a function expression?S**

| **Function Declaration** | **Function Expression** |
| --- | --- |
| A function declaration must have a function name. | A function expression is similar to a function declaration without the function name. |
| Function declaration does not require a variable assignment. | Function expressions can be stored in a variable assignment. |
| These are executed before any other code. | Function expressions load and execute only when the program interpreter reaches the line of code. |
| The function in function declaration can be accessed before and after the function definition. | The function in function expression can be accessed only after the function definition. |
| Function declarations are hoisted | Function expressions are not hoisted |
| Syntax: function geeksforGeeks(paramA, paramB) { // Set of statements } | Syntax: var geeksforGeeks= function(paramA, paramB) { // Set of statements } |

**Q:-3Discuss the concept of parameters and return values in functions.**

**Parameters in Functions:-**

Parameters are variables listed inside the parentheses in a function

**definition.** They act as placeholders for values (called arguments) that are passed to the function when it is called.  
Parameters make functions flexible, as they allow the same function to operate on different data without rewriting the code.  
If no argument is provided for a parameter, its value becomes undefined unless a default parameter value is set.

**Example:**

**function greet(name) {**

**console.log("Hello, " + name);**

**}**

**Return Values in Functions:-**

A return value is the output that a function sends back to the code that called it, using the return statement.  
The return statement ends the function execution and specifies the value to be returned.  
If no return statement is used, the function returns undefined by default.  
Return values are important for storing results, performing calculations, or passing data between functions.

**Example:**

**function add(a, b) {**

**return a + b;**

**}**

**Q:-1 What is an array in JavaScript? How do you declare and initialize an array?**

**Array in JavaScript:-**

An array in JavaScript is a special type of object used to store multiple values in a single variable.  
These values can be of any data type (numbers, strings, objects, even other arrays) and are stored in an ordered list with index numbers starting from 0.  
Arrays make it easier to manage collections of related data.

**Declaring and Initializing an Array:-**

**1. Using Array Literal (Most Common)**

let fruits = ["Apple", "Banana", "Cherry"];

**2. Using the new Array() Constructor**

let numbers = new Array(1, 2, 3, 4);

**3. Empty Array then Adding Elements**

let colors = [];

colors[0] = "Red";

colors[1] = "Blue";

**Q:-2Explain the methods push(), pop(), shift(), and unshift() used in arrays.**

**1. push()**

* Purpose: Adds one or more elements to the end of the array.
* Changes Original Array? yes
* Returns: The new length of the array**.**

**Example:**

**let fruits = ["Apple", "Banana"];**

**fruits.push("Mango");**

**console.log(fruits); // ["Apple", "Banana", "Mango"]**

**2. pop()**

* Purpose: Removes the last element from the array.
* Changes Original Array? yes
* Returns: The removed element.

**Example:**

**let fruits = ["Apple", "Banana", "Mango"];**

**let lastFruit = fruits.pop();**

**console.log(lastFruit); // Mango**

**console.log(fruits); // ["Apple", "Banana"]**

**3. shift()**

* Purpose: Removes the first element from the array.
* Changes Original Array? Yes
* Returns: The removed element.

**Example:**

**let fruits = ["Apple", "Banana", "Mango"];**

**let firstFruit = fruits.shift();**

**console.log(firstFruit); // Apple**

**console.log(fruits); // ["Banana", "Mango"]**

**4. unshift()**

* Purpose: Adds one or more elements to the beginning of the array.
* Changes Original Array? Yes
* Returns: The new length of the array.

**Example:**

**let fruits = ["Banana", "Mango"];**

**fruits.unshift("Apple");**

**console.log(fruits); // ["Apple", "Banana", "Mango"]**

**Objects**

**Q:-1 What is an object in JavaScript? How are objects different from arrays?**

**Object in JavaScript:-**

An object in JavaScript is a data structure used to store data in the form of key–value pairs.

* Keys (also called properties) are strings or symbols.
* Values can be of any data type (numbers, strings, arrays, other objects, functions, etc.).
* Objects are useful for representing real-world entities and structured data.

**Example:**

**let person = {**

**name: "Alice",**

**age: 25,**

**isStudent: true**

**};**

**You can access object properties:**

**console.log(person.name); // Dot notation → Alice**

**console.log(person["age"]); // Bracket notation → 25**

**Difference Between Objects and Arrays:-**

| **Array** | **objects** |
| --- | --- |
| Arrays are best to use when the elements are numbers. | Objects are best to use when the elements' strings (text). |
| The data inside an array is known as Elements. | The data inside objects are known as Properties which consists of a key and a value. |
| The elements can be manipulated using []. | The properties can be manipulated using both.DOT notation and []. |
| The elements can be popped out of an array using the pop() function. | The keys or properties can be deleted by using the delete keyword. |
| Iterating through an array is possible using For loop, For..in, For..of, and ForEach(). | Iterating through an array of objects is possible using For..in, For..of, and ForEach(). |

**Q:-2 Explain how to access and update object properties using dot notation and bracket notation.**

**Accessing and Updating Object Properties in JavaScript:-**

JavaScript objects store data as key–value pairs, and you can access or update these values in two ways: dot notation and bracket notation.

**1. Dot Notation (object.property)**

* **Syntax:**

**objectName.propertyName**

* **Rules:**
  + Property name must be a valid identifier (no spaces, can’t start with a number, only letters, digits, \_, $ allowed).
* **Example:**

**let person = { name: "Alice", age: 25 };**

**// Access**

**console.log(person.name); // Alice**

**// Update**

**person.age = 26;**

**console.log(person.age); // 26**

**2. Bracket Notation (object["property"])**

* **Syntax:**

**objectName["propertyName"]**

* **Rules:**
  + Property name must be a string or variable containing a string.
  + Useful when property names have spaces, special characters, or are stored in variables.
* **Example:**

**let person = { name: "Alice", "favorite color": "blue" };**

**// Access**

**console.log(person["favorite color"]); // blue**

**// Update**

**person["favorite color"] = "green";**

**console.log(person["favorite color"]); // green**

**JavaScript Events**

**Q:-1What are JavaScript events? Explain the role of event listeners**

JavaScript events are actions or occurrences that happen in a web browser, either triggered by user interaction or by the browser itself. These events serve as signals that JavaScript code can respond to, enabling the creation of dynamic and interactive web pages. Examples of common events include:

* **User Interactions:**Clicking a button, typing in an input field, hovering over an element, submitting a form, or pressing a key.
* **Browser Events**: A page finishing loading, an image failing to load, or a window resizing**.**

**Role of Event Listeners:**

Event listeners are a fundamental mechanism in JavaScript for handling these events. An event listener is a function that "listens" for a specific event to occur on a particular HTML element. When that event happens, the associated function (known as the event handler) is executed.

The primary method for attaching event listeners in modern JavaScript is addEventListener(). This method allows developers to:

* Specify the event type: Such as "click", "keydown", "mouseover", or "load".
* Define the target element: The HTML element on which the event is expected to occur.
* Provide the event handler function: The JavaScript code to be executed when the event is detected.

Event listeners provide a flexible and robust way to manage event handling, allowing multiple functions to be attached to the same event on an element without overwriting previous handlers, and offering the ability to easily remove listeners when they are no longerneededusing removeEventListener(). This approach promotes cleaner and more maintainable code compared to older methods like inline event handlers in HTML.

**Q:-2 How does the addEventListener() method work in JavaScript? Provide an example.**

The addEventListener() method in JavaScript is used to attach an event handler to a specified element in the Document Object Model (DOM). This method allows you to listen for specific events, such as a "click" or "keydown", on an element and execute a function when that event occurs.

**How it works:**

* **Select the element:** First, you need to get a reference to the HTML element you want to attach the event listener to. This is typically done using methods like document.getElementById(), document.querySelector(), or document.querySelectorAll().
* **Call addEventListener():** On the selected element, you then call the addEventListener() method.
* **Provide arguments:** This method takes at least two arguments:
  + event (string): The type of event to listen for (e.g., "click", "mouseover", "submit", "keydown"). The "on" prefix is omitted (e.g., use "click" not "onclick").
  + function (function): The callback function to be executed when the specified event occurs. This function receives an Event object as its first parameter, which contains information about the event.
  + useCapture (boolean, optional): An optional third parameter that determines whether the event is handled during the capturing phase or the bubbling phase. The default value is false, which means event bubbling (from the target element up to the document). Setting it to true enables event capturing (from the document down to the target element).

**Example:**

This example demonstrates how to add a "click" event listener to a button, which then changes the text content of a paragraph when clicked.

**<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>addEventListener Example</title>  
</head>  
<body>  
 <button id="myButton">Click Me</button>  
 <p id="myParagraph">Initial text.</p>  
  
 <script>  
 // Get references to the HTML elements  
 const button = document.getElementById('myButton');  
 const paragraph = document.getElementById('myParagraph');  
  
 // Define the function to be executed when the button is clicked  
 function changeParagraphText() {  
 paragraph.textContent = 'Button was clicked!';  
 }  
  
 // Attach the event listener to the button  
 button.addEventListener('click', changeParagraphText);  
 </script>  
</body>  
</html>**

**DOM Manipulation**

**Q:-1 What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?**

**Document Object Model (DOM) in JavaScript**

The DOM is a programming interface for HTML and XML documents.  
It represents the web page structure as a tree of nodes (elements, attributes, text).

* **Document:- The entire HTML document.**
* **Object :- Each element (like <p>, <div>, <h1>) is represented as an object.**
* **Model :- The structured tree-like representation of the page.**

**In simple terms:**DOM = HTML code turned into a JavaScript-accessible tree structure when a web page loads.

**JavaScript and the DOM:-**

**JavaScript can interact with the DOM to:**

1. Access elements – Find elements by ID, class, tag, etc.
2. Change content – Modify text, HTML, and attributes.
3. Change styles – Apply or update CSS properties.
4. Add/Remove elements – Create new nodes or delete existing ones.
5. Handle events – Respond to clicks, inputs, mouse movements, etc.

**Example:**

**JavaScript Interacting with the DOM**

**<p id="demo">Hello World!</p>**

**<button onclick="changeText()">Click Me</button>**

**<script>**

**function changeText() {**

**document.getElementById("demo").textContent = "Hello JavaScript!";**

**}**

**</script>**

**Q:-2 Explain the methods getElementById(), getElementsByClassName(),and querySelector() used to select elements from the DOM.**

**DOM Element Selection Methods in JavaScript**

JavaScript provides several methods to find and select elements from the Document Object Model (DOM).  
Three commonly used methods are getElementById(), getElementsByClassName(), and querySelector().

**1. getElementById()**

* **Purpose: S**elects a single element based on its id attribute.
* **Returns:** The element object, or null if not found.

**Syntax:**

**document.getElementById("idName");**

**Example:**

**let heading = document.getElementById("mainTitle");**

**heading.textContent = "Updated Title";**

**2. getElementsByClassName()**

* **Purpose:** Selects all elements with the specified class name.
* **Returns:** A live HTMLCollection (similar to an array, but not exactly).

**Syntax:**

**document.getElementsByClassName("className");**

**Example:**

**let items = document.getElementsByClassName("list-item");**

**items[0].style.color = "red"; // Changes first matching element**

**3. querySelector()**

* **Purpose:** Selects the first element that matches a given CSS selector.
* **Returns:** The element object, or null if no match is found.

**Syntax:**

**document.querySelector("CSS\_Selector");**

**Example:**

**let firstParagraph = document.querySelector("p.intro");**

**firstParagraph.style.fontSize = "20px";**

**JavaScript Timing Events (setTimeout, setInterval)**

**Q:1 Explain the setTimeout() and setInterval() functions in JavaScript. Howare they used for timing events?**

**setTimeout() and setInterval() in JavaScript**

JavaScript provides timing functions to execute code after a delay or repeatedly at intervals.  
The two most common methods are setTimeout() and setInterval().

**1. setTimeout()**

* **Purpose:** Executes a function once after a specified delay (in milliseconds).

**Syntax:**

**setTimeout(function, delay, param1, param2, ...);**

* **Parameters:**
  + function → Function to execute.
  + delay → Time in milliseconds before execution.
  + Optional parameters → Passed to the function when executed.

**Example:**

**setTimeout(function() {**

**console.log("Hello after 2 seconds");**

**}, 2000);**

**Cancel: Use clearTimeout(timeoutID) to stop it before execution.**

**2. setInterval()**

* **Purpose:** Executes a function repeatedly at specified time intervals (in milliseconds).

**Syntax:**

**setInterval(function, delay, param1, param2, ...);**

* **Parameters:**
  + function → Function to execute repeatedly.
  + delay → Interval in milliseconds between executions.
  + Optional parameters → Passed each time function executes.

**Example:**

**let counter = 0;**

**let intervalId = setInterval(function() {**

**counter++;**

**console.log("Count: " + counter);**

**if (counter === 5) {**

**clearInterval(intervalId); // Stop after 5 times**

**}**

**}, 1000);**

* **Cancel**: Use clearInterval(intervalID) to stop it.

**Q:-2 Provide an example of how to use setTimeout() to delay an action by 2 seconds.**

**Example:** Delay an Action by 2 Seconds using setTimeout()

console.log("Action will start in 2 seconds...");

setTimeout(function() {

console.log("This message appears after 2 seconds!");

}, 2000); // 2000 milliseconds = 2 seconds

**JavaScript Error Handling**

**Q:-1 What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.**

**Error Handling in JavaScript**

Error handling in JavaScript is the process of detecting, catching, and managing runtime errors so the program can continue running without crashing.  
JavaScript provides the try...catch...finally construct to handle errors gracefully.

**1. try Block**

* Contains code that might throw an error.
* If an error occurs, control is passed to the catch block.

**2. catch Block**

* Contains code to handle the error.
* Receives the error object as a parameter, which contains details about the error.

**3. finally Block**

* Contains code that runs after try and catch, regardless of whether an error occurred.
* Useful for cleanup tasks (closing files, stopping timers, etc.).

**Syntax**

**try {**

**// Code that may throw an error**

**} catch (error) {**

**// Code to handle the error**

**} finally {**

**// Code that runs no matter what**

**}**

**Example**

**try {**

**let num = 10;**

**console.log(num / 2); // Works fine**

**console.log(undeclaredVar);**

**}**

**catch (error) {**

**console.log("An error occurred: " + error.message);**

**}**

**finally {**

**console.log("Execution completed (with or without error).");**

**}**

**Q:-2 Why is error handling important in JavaScript applications?**

**Importance of Error Handling in JavaScript Applications**

Error handling is crucial in JavaScript because it ensures that the application can detect, manage, and recover from errors without crashing or producing unexpected results.

**Reasons Why Error Handling is Important**

1. **Prevents Application Crashes**
   * Without error handling, a single runtime error can stop the entire script from running.
   * Error handling allows the program to continue executing other parts of the code.
2. **Improves User Experience**
   * Instead of showing a blank page or breaking functionality, you can display a friendly error message to the user.
3. **Helps with Debugging**
   * Capturing errors provides detailed information (error message, type, stack trace) that developers can use to fix bugs.
4. **Graceful Recovery**
   * Allows the application to switch to alternative logic when something fails (e.g., using fallback data when an API request fails).
5. **Security and Stability**
   * Proper error handling prevents sensitive system information from being exposed to the user.
   * It ensures the application behaves predictably even in unexpected situations.
6. **Maintains Application Flow**
   * With error handling, only the problematic part of the code stops, not the entire application.