Module 17) Javascript For Full Stack Assignment

1. JavaScript Introduction

Theory Assignment`

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

Ans: JavaScript is a high-level, interpreted programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. It enables interactive web pages and is an essential part of web applications.

- **Client-side (primarily)**: Runs in the browser, but can also run on servers using environments like **Node.js**.
- **Dynamically typed** and **event-driven**.
- Supports **object-oriented**, **functional**, and **procedural** programming styles.

❖ **Role of JavaScript in Web Development**

1. **Dynamic Content Updates**
   - Change content on the page without reloading.
2. **User Interaction**
   - Responds to user actions like clicks, mouse movements, form inputs.
3. **Asynchronous Communication**
   - Uses technologies like AJAX or Fetch API to request data from servers in the background.
4. **Browser and DOM Manipulation**
   - Access and modify the Document Object Model (DOM).
5. **Animations and Effects**
   - Create smooth UI animations and transitions.
6. **Storage and Cookies**

- tore data locally using Web Storage API (localStorage, sessionStorage) or cookies.

7. **Frameworks and Libraries**
   - Use of libraries (e.g., jQuery) and frameworks (e.g., React, Angular, Vue) to build complex applications efficiently.

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

Ans: different from other programming languages like Python or Java

Features	Javascript	Python	Java
Primary Use	Web development (front-end & back-end)	General-purpose (web, data science, automation, AI)	General-purpose, especially enterprise apps & Android
Execution Environment	Runs in browsers; Node.js for server-side	Runs on Python interpreter	Runs on Java Virtual Machine (JVM)
Syntax	like less strict, uses curly braces {}	English-like, focuses on readability, indentation-based	like, verbose, strict typing
Typing	Dynamically typed	Dynamically typed	Statically typed
Compiled/Interpreted	Interpreted (JIT compiled in modern engines)	Interpreted	Compiled to bytecode (then run on JVM)
Concurrency	Event-driven, non-blocking (asynchronous)	Multi-threading, asyncio	Multi-threading

Object-Oriented	Yes (prototype-based)	Yes (class-based, supports multiple paradigms)	Yes (strictly class-based OOP)
Learning Curve	Moderate (because of quirks & async)	Beginner-friendly	Steeper (due to verbosity and boilerplate)
Popular Use Cases	Interactive websites, SPAs, web servers	AI/ML, automation, scripting, web apps	Android apps, enterprise systems, desktop apps

3: Discuss the use of <script> tag in HTML. How can you link an external JavaScript?.

Ans: The <script> tag in HTML is used to embed or reference JavaScript code within an HTML document. JavaScript is a scripting language that enables dynamic content, interactivity, and enhanced functionality on websites.

- **There are Two Define a Javascript**
  1. **Inline Javascript Using <script>Tag**
     - You can write JavaScript code directly within the HTML file using the <script> tag.
     - Syntax:- <script>…… </script>
     - Example:- <!DOCTYPE html>

```
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Connect External File</title>
</head>
<body>
<script>
```

```
let no1=200;
 let no2=20;
  alert(`Addition=> ${no1+no2}`)
  alert(`Subtraction=>${no1-no2}`)
  alert(`Multification=> ${no1*no2}`)
  alert(`Divison=> ${no1/no2}`)
   alert(`Modulo ${no1%no2}`)

 document.writeln("Addition:",no1+no2,"<br>")
  document.writeln("Subtration:",no1-no2,"<br>")
 document.writeln("Multification:",no1*no2,"<br>")
 document.writeln("Division:",no1/no2,"<br>")
  document.writeln("Modulo:",no1%no2,"<br>")
</script>
</body>
</html>
```

2. **Linking an External JavaScript File**
   - To keep your code modular and easier to maintain, it's common practice to link to an external JavaScript file using the src attribute of the <script> tag.
   - Syntax: <script src="./demo/index.js"></script>
   - **Example:**

```
<!DOCTYPE html>
<html>
   <head>
  <title>External Script File</title>
  </head>
<body>
```

```
<h1>External Script Demo</h1>
<!—Link to  external JavaScript file -->
<script src="./demo/index.js”></script>
</body>
</html>
```

1. Variables and Data Types

Theory Assignment

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

Ans: Variables in JavaScript are containers for storing data values.

- **There are Two types of Variables**
  1. **Global Variable**
     - A JavaScript global variable is declared outside the function or declared with window object. It can be accessed from any function.
  2. **Local Variable**
     - A JavaScript local variable is declared inside block or function. It is accessible within the function or block only.
- **How do you declare a variable**
  1. **var:-var is a global variable.**

     Syntax:- var variable_name=value/data
- **Example:- var no=10 // Declaration are allow**

  Var no=20 // Re-declaration are allow

  no=30 // assignment are allow

  document.write(“Number=>”,no);

  2. **let:- let is local variable,(scope variable)**

     Syntax:- let variable_name=data/value
- **Example:- let no2=10 // Declaration are allow**

  let no2=20 // Re-declation are not allow

no2=30 // Re-assignment are allow

document.write(“No2:”,no2)

3. const: To store a unique data/value.

Syntax:- const variable_name=data/value

- **Example:- const no3=3.14 // Declation are allow**

  const no3=5.11 // Re-declation are not allow

  no3=6.12 // Re-assignment are not allow

  document.write(“No3=>”,no3)

2: Explain the different data types in JavaScript. Provide examples for each.

Ans: JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type

- This data type is in-built provide in javascript.
- There are many Types of Primitive Data types.
  - **Number:-** Represents numeric values(e.g. 0 -100)
    - Syntax:- datatype variable_name=data/value
- Example: var no1=10, no2=20

```
console.log(typeof(no1));
console.log(“Additon=>”,no1+no2);
console.log(“Subtraction=>”,no1-no2);
console.log(“Multification=>”,no1*no2);
console.log(“Division=>”,no1/no2);
```

  - **String:-** To store a sequence of characters(e.g.- “Hello World”). To store single quotes(‘ ’) and double quotes(“ ”).

    Syntax:- datatype variable_name = character

    Example:- string data=”Rahul Parmar”

    console.log(typeof(data))

    console.log(data)

document.write(data)

- **Boolean:-** Represents boolean value either false or true
  Syntax:- datatype variable_name=true/false
  Example:- boolean vel=true;
  boolean wel=false;
  console.log(typeof(wel))
  console.log(vel);
  console.log(wel);
- **Undefined:-** Represents undefined value
  Syntax:- datatype variable_name
  Example:- let x;
  console.log(x);
  console.log(typeof(x));
- **Null:-** To Store a null value (Empty value)
  Syntax:- datatype variable_name=null
  Example:- let gn=null;
  console.log(gn);
  console.log(typeof(gn));
- **Bigint**:- to print a large amount of values
  Syntax:- datatype variable_name = BigInt(values)
  Example:- let data=BigInt(1,2,3,4,5,6,7,8,9,10)
  console.log(data)
  console.log(typeof(data))
- **Symbol:-** to print a symbol
  Syntax:- datatype variable_name=Symbol(symbols)
  Example: let d1=Symbol("$$");
  Console.log(d1);
  Console.log(typeof(d1));

2. Non-primitive data type

- Non-primitive data types in JavaScript, also known as reference types, To store a large number value in a single variable. This is user define data type.
- There are two types of non-primitive data types in javascript

1. **Array:-** Array is a collection of multiple data to store a single variable. Array always access to values in index. Index always start 0; Array is define than used a Square braket [].

   Syntax:- datatype variable_name =[primitive];

   Example:- let no1=[10,20,30,40,50,60,70]

```
console.log(no1);
console.log(typeof(no1))
console.log(no1[0])
console.log(no1[1])
console.log(no1[2])
console.log(no1[3])
```

2. **Object:-** Object is collection of multiple data to store in single variable using curly braces {}. Object work key/pair and value than access data/values.

   Syntax:- datatype variable_name = {key:value}

   Example:- let company = {emp_id: 1,

```
name: “Mukesh Kumar”,
designation: “Software Engineer,
salary: 45000
}

console.log(company)

console.log(typeof(company))

console.log(company.id);

console.log(company.name)

console.log(company.designation)
```

3: What is the difference between undefined and null in JavaScript?

Ans: Difference between undefined and null in Javascript

Undefined	Null

It represents the missing or uninitialized value.	**It represents the intentional absence of a value.**
It is automatically assigned to a variable or object that is not initialized.	**It is an assignable value.**
It was introduced in **ECMAScript1** (ES1).	It is a primitive value in Js.
typeof operator returns 'undefined.'	typeof operator returns 'object.'
While performing arithmetic operations, undefined is converted to NaN.	During arithmetic operations, null is converted to Zero(0).

3. JavaScript Operators

Theory Assignment

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

1. **Arithmetic operators**
2. **Assignment operators**
3. **Comparison operators**
4. **Logical operators**

Ans: JavaScript operators are symbols that are used to perform operations on operands. There are several types of operators

- **Arithmatic operartors**
  - **Arithmetic operators are used to perform arithmetic operations on the operands.**
  - **Addition(+), Subtraction(-), Multification(*), Division(/), Modulo(%).**
    - **Example:-** <!DOCTYPE html>
      <html lang="en">

```
    <head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Arithmatic Operations</title>
</head>
    <body>
     <script>
            let no1=400;
             let no2=20;
            document.writeln("Addition:",no1+no2,"<br>")
             document.writeln("Subtration:",no1-no2,"<br>")
            document.writeln("Multification:",no1*no2,"<br>")
            document.writeln("Division:",no1/no2,"<br>")
             document.writeln("Modulo:",no1%no2,"<br>")
</script>
</body>
</html>
```

- **Assignment operators**
  - **Assignment operator to assign a value in a variable.**
  - **Example:- a=100, b=50**

Operator	Description	Example
=	Assign	a = b
+=	Add and Assign	a +=b
-=	Subtract and Assign	a -= b
*=	Multiply and Assign	a *= b

/=	Divide and Assign	a /= b
%=	Modulus and Assign	a %= b

➢ Comparison operators

- The JavaScript comparison operator compares the two operands.
- Example:- let a=10, b="10";

Operator	Description	Example	Result
==	Equal to	a == b	True
===	Equal to (type + value)	a===b	False
!=	Not equal to	a != b	False
!==	Strictly not equal	a!==b	True
>	Greater than	10>2	True
<	Less than	10<2	False
>=	Greater than or equal to	10>=10	True
<=	Less than or equal to	10<=5	False

➢ Logical operators

- **Logical operators are used to determine the logic between variables or values.**

Operator	Description	Example
&&	Logical AND	a && b
||	Logical OR	a || b
!	Logical Not	!a

- **Example:- let a = 10, b = 5;**
  console.log(a > 5 && b < 10); // true

```
console.log(a < 5 || b < 10); // true
console.log(!(a === 10));     // false
```

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

Ans: Difference Between == and === in JavaScript

	== (Double Equals)	=== (Triple Equals)
Check	Only Check the value	Check the value and data type.
Performance	Slower (due to conversion)	Faster (no conversion needed)
Use Case	Loose comparison (if type conversion is acceptable)	Strict comparison (when type must match)

- **Example:- let a=10, b='10';**

```
let d = a == b;
console.log(d)  // Output:- True

let w= a===b;

console.log(w) // Output: False
```

4. Control Flow (If-Else, Switch)

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

Ans: The JavaScript Conditional statement is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

- **If Statement**
  - ✓ It evaluates the content only if expression is true.
  - ✓ Syntax: if(expression){ //content to be evaluated }
  - ✓ Example: let age=17; if(age>=16){ console.log("Age is valid"); }
- **If else statement**

- ✓ It evaluates the content whether condition is true of false.
- ✓ Syntax: if(expression){ //evaluated if condition is true } else{ //evaluated if condition is false }
- ✓ Example:- let no=10; if(no%2==0) { console.log(“Even Number”) } else { console.log(“Odd Number”) }

➢ if else if statement

- ✓ It evaluates the content only if expression is true from several expressions.
- ✓ Syntax: if(expression1){ //content to be evaluated if expression1 is true } else if(expression2){ //content to be evaluated if expression2 is true } else{ //content to be evaluated if no expression is true }
- ✓ Example: let marks=59

```
if(marks>100 || marks<0) { console.log("Not Valid marks") }
else if(marks>= 90 && marks <=100) { console.log("Grade A") } else if(marks>=70 && marks<=90) { console.log("Grade B") } else if(marks>=60 && marks<=70) { console.log("Grade C") }
else { console.log("Fail") }
```

➢ nested if else

- ✓ You can nest if else statements with other if else statements, creating conditions at multiple levels. Nested if else statements allow for more complex decision-making within the program.
- ✓ Syntax:

```
if (condition1) {
// Code block for condition1 being true
if (condition2) {
// Code block for condition1 and condition2 both being true
} else {
// Code block for condition1 being true and condition2 being false
}
} else {// Code block for condition1 being false
}
```

- ✓ Example: let no=20;
- ✓ if(no>0) { console.log(“Positive Number:”);
  if(no%2==0 {
  console.log(“Even Number:”); }
  }else{
  console.log(“Odd Number:”);} }
  else { console.log(“Negative Number:”); } }

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

Ans: The JavaScript switch statement is used to execute one code from multiple expressions.

- ✓ **Syntax: switch(expression){**
  case value1: //code to be executed; break;
  case value2: //code to be executed; break; .....
  default: //code to be executed if above values are not matched; }
- ✓ **Example: let fruit = "apple";**
  switch(fruit) {
  case "banana": console.log("Yellow fruit"); break;
  case "apple": console.log("Red or green fruit"); break;
  case "orange": console.log("Orange fruit"); break;
  default: console.log("Unknown fruit"); }
  - ➢ **<u>When to Use switch Over if-else:</u>**
    - **Use switch when:**
      - You are checking the same expression or variable against multiple constant values.
      - You have a long chain of if-else if statements that compare the same variable to different values.
      - You want cleaner and more readable code for multiple discrete value checks.
    - **Use if-else when:**
      - You are evaluating different expressions.
      - You are doing range comparisons or more complex conditions.

- You want to use logical operators (e.g., &&, ||, >, <).

5. Loops (For, While, Do-While)

Theory Assignment

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

Ans: JavaScript provides several types of loops that allow you to execute a block of code repeatedly under certain conditions. There are several types of loops.

- **for Loop**
  - Used when the number of iterations is known or can be controlled by a counter.
  - **Syntax:** for (initialization; condition; increment) {
    // code to execute }
  - **Example let no = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**
    for (let i = 0; i < no.length; i++) {
    console.log("Number=>", no[i]); }

    // Output: 1,2,3,4,5,6,7,8,9,10

- **while Loop**
  - Used when the number of iterations is unknown, and you want to loop while a condition is true.
  - **Syntax**: while (condition) { // code to execute }
  - **Example**: **let i = 0; while (i <= 10) {**

    console.log(“Number”, i) i++; } // Output: 0,1,2,3,4,5,6,7,8,9,10

- **do-while Loop**
  - Similar to the while loop, but the code block executes at least once, even if the condition is initially false.
  - **Syntax:** do { // code to execute } while (condition);
  - **Example:** do{ console.log("Number",i) i++; } while(i<=10); **// Output: 0,1,2,3,4,5,6,7,8,9,10**

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

Ans: **Difference Between while loop and do-while loop**

while Loop	do-while
Condition is checked first then statement(s) is executed.	Statement(s) is executed atleast once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	Semicolon at the end of while. while(condition);
Variable in condition is initialized before the execution of loop.	variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.
while(condition) { statement(s); }	do { statement(s); } while(condition);

6. Functions

Theory Assignment

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

Ans: **Functions** in JavaScript are **reusable blocks of code** that perform a specific task. You can define a function once and use (or "call") it multiple times throughout your program.

- **Declaring a Function without Parameter Syntax:**

```
function functionName() {
  // code to execute
}
```

  - **Example:**

```
function data()  {
        let no1=10,no2=20;
```

```
        console.log("Addition:=>",no1+no2);
        console.log("Subtraction:=>",no1-no2);
        console.log("Multification:=>",no1*no2);
    console.log("Division:=>",no1/no2);        }
```

- **Calling a Function without Parameter Syntax:**
  function_name();
  - **Example: data();**

✓ **Declaring a Function with Parameter Syntax:**

function functionName(parameter) { // code to execute }

- **Example: function data1(no1,no2) {**

```
        console.log("Addition:",no1+no2);
        console.log("Subtraction:",no1-no2);
        console.log("Multification:",no1*no2);
        console.log("Division:",no1/no2);  }
```

- **Calling a Function with Parameter Syntax:**
  function_name(arguments);
- **Example: data1(60,5);**

2: What is the difference between a function declaration and a function expression?

Ans: <u>Difference Between Function declaration and Function expression.</u>

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 data(no1, no2) { // Set of statements }	**Syntax:** var data= function(no1, no2) { // Set of statements }

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

Ans: In JavaScript, parameters and return values are essential for making functions flexible and reusable.

- **Parameters**
  - Parameters are variables listed inside the function definition. They act as placeholders for the values (arguments) that you pass into the function when calling it.
  - **Syntax:** function functionName(parameter) { // code to execute}
  - **Example:** function student(name) {

    console.log("Name=>",name) }

    student(“Rahul Parmar”);

- **Return Values**
  - A return value is the output a function sends back to the part of the program that called it, using the return statement.
  - **Syntax: function function_name() { return value }**
  - **Example: function add() { let no1=20,no2=10;**

    let sum=no1+no2; return sum; }

    function sub() { let no1=20,no2=10;

    let sub=no1-no2; return sub }

    function multi() { let no1=20,no2=10;

    let multi=no1*no2; return multi }

```
function div()  { let no1=20,no2=10; let div=no1/no2;
return div }

console.log("Addition=>",add())
console.log("Subtraction=>",sub())
console.log("Multification=>",multi())
console.log("Division=>",div())
```

7. Arrays

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

Ans: Array is a collection of multiple data to store a single variable. Array always access to values in index. Index always start 0; Array is define than used a Square braket [].

- **Syntax:** datatype variable_name =[primitive];
- **Example:** let data=[10,20,30,40,50,60,70,80,90,100] console.log(data)

```
console.log(data.length)
document.writeln(data,"<br>")
for(let i=0;i<data.length;i++)
{
  console.log("Index=>",i,"Value=>",data[i])
  document.writeln("Index=>",i,"Value=>",data[i], "<br>")
}
```

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

Ans: In JavaScript arrays come with built-in methods to add or remove elements.

- **push() method**
  - Add one or more elements to the end of an array.
  - **Syntax:** push(element);
  - **Example**: let data = [10, 20, 30, 40, 50, 60, 70]

data.push(80) console.log("Add data=>",data) **// Output: [10,20,30,40,50,60,70,80]**

- **pop() method**
  - Remove the last element from an array.
  - **Syntax:** pop(element)
  - **Example:** let data = [10, 20, 30, 40, 50, 60, 70]

    data.pop(70) console.log("Remove data=>",data) **// Output: [10,20,30,40,50,60]**

- **shift() method**
  - Removes the first element from an array.
  - **Syntax:** shift()
  - **Example:** let data = [10, 20, 30, 40, 50, 60, 70]
    data.shift();
    console.log("Remove First Data=>",data) **// Output: [20,30,40,50,60,70]**
- **unshift() method**
  - Add one or more elements to the beginning of an array.
  - **Syntax: unshift(element)**
  - **Example:** let data = [10, 20, 30, 40, 50, 60, 70]
    data.unshift(9)
    console.log("Add First Data=>",data) **// Output: [9,10,20,30,40,50,60,70]**

8. Objects

Theory Assignment

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

Ans: Object is collection of multiple data to store in single variable using curly braces {}. Object work key/pair and value than access data/values. this is user define index(key), always index(key) start alfa numeric.

object property access using dot .

- **Syntax:-** datatype variable_name = {key:value}
- **Example:** let Bank = {

```
Bank_name : "SBI Bank",
Account_holder_name: "Mukesh Patel",
Age : 25,
Designation : "Software Engineer",
Palace : "Ahmedabad"    }
    console.log(Bank)
    console.log(Bank.Bank_name)
    console.log(Bank.Account_holder_name)
    console.log(Bank.Age)
    console.log(Bank.Designation)
    console.log(Bank.Palace)
```

- **How Are Objects Different from Arrays?**

Feature	Objects	Arrays
Purpose	**Store named values (key-value pairs).**	**Store ordered collections of items (indexed).**
Key Type	Keys are **strings** (or Symbols).	Keys are **numeric indexes** (starting from 0).
Use Case	When data is structured as properties.	When data is a list or sequence.
Example Syntax	{ name: "Alice", age: 30 }	**["Alice", 30]**
Access Method	object.key or object["key"]	array[index]
Iteration	for...in, Object.keys(), Object.entries()	for, forEach(), map(), filter(), etc.

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

Ans: In JavaScript you can access and update object properties using two main syntaxes: dot notation and bracket notation.

- **Dot notation**
  - **Accessing a propert**
    - let company ={Id: 15,
      name: "Ganesh Bhai",
      designation : "Sales manager",
      salary: 45000
      }
      console.log(company)
      console.log(company.name)
      console.log(company.salary)
  - **Updating a property**
    - company.salary= 65000 // change data

- **Bracket Notation**
  - The property name is dynamic (stored in a variable). The property name contains special characters, spaces, or is not a valid identifier.
  - **Example:** let person = { "full name": "Ramesh Sharma", age: 30 };
  - **// Access**
  - console.log(person["full name"]); **// Output: "Ramesh Sharma**
  - **// Update**
  - person["age"] = 32;
  - console.log(person["age"]); **// Output: 32**

9. JavaScript Events

Theory Assignment

1: What are JavaScript events? Explain the role of event listeners.

Ans: In JavaScript, events are actions or occurrences that happen in the browser, usually as a result of user interaction or the browser itself. These can include things. JavaScript supports a variety of event types. Common categories include:

- **Mouse Events**: click, dblclick, mousemove, mouseover, mouseout
- **Keyboard Events:** keydown, keypress, keyup

- **Form Events:** submit, change, focus, blur
- **Window Events:** load, resize, scroll

❖ Common JavaScript Events

Event Attribute	Description
onclick	Triggered when an element is clicked.
onmouseover	Fired when the mouse pointer moves over an element.
onmouseout	Occurs when the mouse pointer leaves an element.
onkeydown	Fired when a key is pressed down.
onkeyup	Fired when a key is released.
onchange	Triggered when the value of an input element changes.
onload	Occurs when a page has finished loading.
onsubmit	Fired when a form is submitted.
onfocus	Occurs when an element gets focus.
onblur	Fired when an element loses focus.

➢ Role of Event Listeners

- An event listener is a function that waits for a specific event to occur on a given element and then executes some code in response.
- **Syntax: element.addEventListener("eventType", callbackFunction);** // eventType: A string like "click", "mouseover", "keydown", etc. callbackFunction: The function to run when the event happens.

➢ Why Use Event Listeners?

- **Separation of concerns**: Keeps JavaScript logic separate from HTML.
- **Reusable and modular**: You can add, remove, or change behavior easily.

- **Dynamic interaction**: Makes it possible to respond to real-time user actions.

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

Ans: The **addEventListener()** method in JavaScript is used to attach an event handler to a specified element. This enables the element to listen for specific events and execute a function in response. You can add multiple event listeners of the same type to the same element. You can also remove them using removeEventListener() if needed. It's more flexible than using onclick or inline event handlers.

- **Syntax**: element.addEventListener(event, callback, useCapture);
  - **event:** The name of the event (as a string), like "click", "keydown", "mouseover".
  - **callback:** The function to execute when the event occurs.
  - **useCapture *(optional)*:** A boolean value indicating whether to use event bubbling (false, default) or capturing (true).
- **Example:** <button id="myButton">Click Me!</button>

```
let button = document.getElementById("myButton");

button.addEventListener("click", function () {
  alert("Button was clicked!");
});
```

10. DOM Manipulation

Theory Assignment

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

Ans: The HTML DOM (Document Object Model) is a programming interface that represents the structure of a web page in a way that programming languages like JavaScript can understand and manipulate.

- It represents the page structure as a tree of objects, where each element, attribute, and piece of text is a node in the tree.

- **How JavaScript Interacts with the DOM**
  - ✓ JavaScript can:
    1) Access elements
    2) Change content or attributes
    3) Change styles
    4) Create or delete elements
    5) **Respond to user interactions** using events
- **Example of Interaction of DOM**

```
<p id="message">Hello World</p>
  <button id="changeBtn">Hello India</button>
    const paragraph = document.getElementById("message");
    const button = document.getElementById("changeBtn");
    button.addEventListener("click", function () {
      paragraph.textContent = "Text changed with JavaScript!";
    });
```

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

Ans: In JavaScript, you use DOM selection methods to access HTML elements so you can manipulate them. There are three commonly used methods.

- **getElementById()**
  - Selects **one element** by its **unique id** attribute. **Returns**: A single **element object** or null if not found.
  - **Syntax: getElementById("id")**
  - **Example:**

```
<html lang="en">  <head>
  <title>getElementById example</title> </head> <body>
  <p id="para">Hello World! </p>
  <button onclick="changeColor('blue');">blue</button>
  <button onclick="changeColor('red');">red</button>
    <script>
    function changeColor(newColor) {
        const elem = document.getElementById("para");
        elem.style.color = newColor; }
```

```
        </script>
    </body>
</html>
```

- **getElementsByClassName()**
  - **Selects all elements with a specific class name.**
  - **Syntax: getElementsByClassName("className")**
  - **Example**: Number 1 Number 2

```
let elements = document.getElementsByClassName("note");
    console.log(elements.length); // Output: Number 1
    console.log(elements[0].textContent); // Output: Number 2
```

- **querySelector()**
  - Selects the first element that matches a CSS selector. Returns: A single element object or null.
  - **Syntax: querySelector(selectors)**
  - **Example:** Hello World!
  - let firstText = document.querySelector(".container .text");
    console.log(firstText.textContent); // Output: Hello World!

11. JavaScript Timing Events (setTimeout, setInterval)

Theory Assignment

1: Explain the setTimeout() and setInterval() functions in JavaScript. How are they used for timing events?

Ans: Both setTimeout() and setInterval() are timing functions used to schedule code execution in JavaScript. They allow you to delay or repeat actions over time, making them useful for animations, reminders, polling, or delayed actions.

- **Asynchronous Execution:** Both methods are asynchronous, meaning the browser doesn't block other code execution while waiting for the timer.
- **Return Value:** Both methods return a unique identifier (ID), which can be used with clearTimeout() or clearInterval() to stop the scheduled task.

- **Infinite Intervals:** Using setInterval() without a clearInterval() call can lead to infinite loops, potentially causing performance issues.
- **Nested Timers:** A setTimeout() can mimic a setInterval() by recursively calling itself after each execution.

➢ setTimeout() function

- Executes a function **once after a specified delay** (in milliseconds). A timeout ID (can be used to cancel the timeout with clearTimeout()).executes a function, after waiting a specified number of milliseconds.
- **Syntax: setTimeout(function, delay)**
- **Example:** console.log(1) console.log(2); console.log(3); console.log(4);

console.log(5); setTimeout(() => { console.log(6); }, 2000); console.log(7); console.log(8); console.log(9); console.log(10);

➢ setInterval() function

- Repeatedly executes a function every milliseconds. An interval ID (can be used to stop the interval with clearInterval()).This method is useful for tasks that need periodic execution, like updating animations or refreshing data.
- **Syntax: setInterval(function, delay);**
- **Example: setInterval(()=>{ console.log("India and Russia Good Friendly Country") },1000)**

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

Ans: <u>setTimeout()</u>

- **Example:** console.log(1)
  console.log(2)
  console.log(3)
  console.log(4)

  console.log(5)

  setTimeout(() => { console.log(6); }, 2000);

```
console.log(7)
console.log(8)
console.log(9)
console.log(10)
```

12. JavaScript Error Handling

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

Ans: **Error handling** in JavaScript is the process of catching and managing runtime errors to prevent a program from crashing. JavaScript provides the try, catch, and finally blocks to handle exceptions gracefully.

- JavaScript uses throw to create custom errors and try...catch to handle them, preventing the program from crashing. The finally block ensures that code runs after error handling, regardless of success or failure.
  - ✓ **throw:** Used to create custom errors and stop code execution
  - ✓ **try...catch:** Allows you to catch and handle errors, preventing the program from crashing.
    - **try block:** Contains code that may throw an error.
    - **catch block:** Catches and handles the error.
  - ✓ **finally:** Executes code after the try and catch blocks, regardless of an error occurring.
  - ✓ **Custom Errors:** You can create your own error types by extending the Error class.
- **Syntax**: **try { // Code that might throw an error**
  } catch (error) {
  // Code to handle the error
  } finally {
  // Code that runs regardless of an error
  }
- **Example: function divide(a, b) {**
  try {
  // Check if b is zero and throw an error if true

```
  if (b === 0) {
     throw new Error("Cannot divide by zero");  }
 let result = a / b;
   console.log(`Result: ${result}`);}
catch (error) {
  console.log(`Error: ${error.message}`);  }
finally {
   console.log("Division attempt completed.");   }  }

// Test cases
divide(10, 2); // Valid division //Output: Result: 5
divide(10, 0); // Triggers error  // Output:  Error: Cannot divide by zero
```

2: Why is error handling important in JavaScript applications?

Ans: Error handling is a crucial part of writing reliable, user-friendly, and maintainable JavaScript applications. Here's why it's important:

- **Prevents Application Crashes**
  - ✓ Without error handling, an unexpected error (e.g., invalid input, network failure, or runtime bug) can cause your entire app or webpage to break or freeze.
  - ✓ **With error handling**, you can catch and manage those errors gracefully to keep the app running.
- **Improves User Experience**
  - ✓ Users shouldn't see cryptic error messages or blank screens. Proper error handling lets you show friendly messages like: “Oops! Something went wrong. Please try again.”
- **Easier Debugging and Logging**
  - ✓ Using try-catch, you can log detailed error information for developers without exposing it to users**. Ex. console.error("Error details:", error);**
- **Supports Resilient Code**
  - ✓ JavaScript apps often interact with:
    - APIs (which might fail)
    - Databases (which might return unexpected data)

- User input (which might be invalid)

✓ Handling these potential issues makes your code **more robust**.

➢ **Essential for Asynchronous Operations**

✓ When using fetch(), async/await, or Promises, errors can happen at any time. Proper handling ensures async failures (like network errors) don't go unnoticed.

✓ **try {**
const response = await fetch('/api/data');
const data = await response.json();}
catch (error) {
console.error("Failed to fetch data:", error); }

➢ **Security**

✓ Uncaught errors might expose sensitive technical information to users or attackers. Handling errors lets you control what gets shown and what stays hidden