Module 17) Javascript For Full Stack Assignment

1. JavaScript Introduction

Theory Assignment

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

JavaScript is a programming language used to make web pages interactive and dynamic. It runs in the web browser and allows users to interact with the website easily.

Role of JavaScript in Web Development:

- It helps to add effects like animations and pop-up messages.
- It can change web page content without reloading the page.
- It is used for checking form inputs before sending them to the server.
- It makes web pages more user-friendly and responsive.

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

JavaScript is different from other programming languages like Python or Java in the following ways:

1. Execution:
    a. JavaScript runs in the web browser (client-side).
    b. Python and Java usually run on the server or computer system.
2. Syntax and Use:
    a. JavaScript is mainly used for web development to make pages interactive.
    b. Python is used for data science, AI, and backend development.
    c. Java is used for mobile apps, enterprise software, and backend systems.
3. Compilation:
    a. JavaScript is interpreted directly by the browser.
    b. Java is compiled before running.

c. Python is interpreted but not mainly used in browsers.

4. Platform:
    a. JavaScript works inside web browsers.
    b. Python and Java need separate environments to run.

Question 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 add JavaScript code to a web page. It helps in making the page dynamic and interactive.

Use of <script> tag:

- It can be used to **write JavaScript directly** inside the HTML document.
- It can also be used to **link an external JavaScript file**.
- The browser reads and executes the JavaScript code inside the <script> tag.

Example (Internal JavaScript):

```
<script>
  alert("Welcome to my website!");
</script>
```

Linking an External JavaScript File:

To use JavaScript from an external file, you use the **src** attribute in the <script> tag.

Example (External JavaScript):

```
<script src="script.js"></script>
```

2. Variables and Data Types

Theory Assignment

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

and const?

In JavaScript, **variables** are used to **store data values** such as numbers, text, or objects. They act like containers that hold information which can be used and changed in a program.

Declaring Variables in JavaScript:

1. **Using var:**
    a. It is the **old** way to declare variables.
    b. Variables declared with var have **function scope** and can be **redeclared**.

```
var name = "Zeel";
```

2. **Using let:**
    a. Introduced in ES6 (modern JavaScript).
    b. Has **block scope** and **cannot be redeclared** in the same block.

```
let age = 20;
```

3. **Using const:**
    a. Also introduced in ES6.
    b. Used to declare **constant variables** whose values **cannot be changed**.

```
const city = "Ahmedabad";
```

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

JavaScript has different **data types** used to store different kinds of values. They are mainly divided into **primitive** and **non-primitive (object)** types.

1. String

Used to store text or characters.

let name = "Zeel";

2. Number

Used to store numeric values (both integer and decimal).

let age = 25;
let price = 99.99;

3. Boolean

Stores only two values: **true** or **false**.

let isOnline = true;

4. Null

Represents an empty or unknown value.

let result = null;

5. Object

Used to store **key-value pairs** or group related data.

let person = { name: "Zeel", age: 20 };

6. Array

A special type of object used to store **multiple values** in a list.

let fruits = ["apple", "banana", "mango"];

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

Feature	undefined	null

Meaning	Variable declared but not assigned a value	Represents intentional absence of value
Type	Undefined type	Object type
Set by	JavaScript automatically	Programmer manually

3. JavaScript Operators

Theory Assignment

Question 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 operations.

Operator	Description	Example	Result
+	Addition	5 + 2	7
-	Subtraction	5 - 2	3
*	Multiplication	5 * 2	10
/	Division	10 / 2	5
%	Modulus (Remainder)	5 % 2	1
**	Exponentiation	2 ** 3	8

2. Assignment Operators:

Used to assign values to variables.

Operator	Description	Example	Same As
=	Assign	x = 5	—
+=	Add and assign	x += 2	x = x + 2
-=	Subtract and assign	x -= 2	x = x - 2

*=	Multiply and assign	x *= 2	x = x * 2
/=	Divide and assign	x /= 2	x = x / 2

3. Comparison Operators:

Used to compare two values and return **true** or **false**.

Operator	Description	Example	Result
==	Equal to	5 == '5'	true
===	Strict equal (checks type)	5 === '5'	false
!=	Not equal	5 != 3	true
>	Greater than	5 > 2	true
<	Less than	3 < 5	true
>=	Greater than or equal	5 >= 5	true
<=	Less than or equal	4 <= 5	true

4. Logical Operators:

Used to combine or invert conditions.

Operator	Description	Example	Result
&&	Logical AND	(5 > 2 && 3 > 1)	true
`		`	Logical OR
!	Logical NOT	!(5 > 2)	false

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

Operator	Name	Description	Example	Result
==	**Equality Operator**	Compares **only values**, not data types. It performs **type conversion**if needed.	5 == '5'	true
===	**Strict Equality Operator**	Compares **both value and data type**. It does **not** perform type conversion.	5 === '5'	false

4. Control Flow (If-Else, Switch)

Theory Assignment

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

Example.

Control flow in JavaScript means the **order in which the code is executed** in a program.
Normally, the code runs **from top to bottom**, but with control statements like if, else if, and else, we can **make decisions**and **change the flow** based on certain conditions.

If-Else Statement:

The if-else statement is used to execute different code blocks depending on whether a condition is **true** or **false**.

Syntax:

```
if (condition) {
  // code runs if condition is true
} else {
  // code runs if condition is false
}
```

Example:

```
let age = 18;

if (age >= 18) {
  console.log("You are eligible to vote.");
} else {
  console.log("You are not eligible to vote.");
}
```

Output:

You are eligible to vote.

Question 2: Describe how switch statements work in JavaScript. When should you use a

switch statement instead of if-else?

A **switch statement** in JavaScript is used to execute different actions based on different conditions for the **same variable or expression**. It checks the value of the variable against multiple cases and runs the matching block of code.

Syntax:

```
switch(expression) {
  case value1:
    // code if expression == value1
    break;
  case value2:
    // code if expression == value2
    break;
  default:
    // code if no case matches
}
```

Example:

```
let grade = 'B';

switch(grade) {
  case 'A':
    console.log("Excellent");
    break;
  case 'B':
    console.log("Good");
    break;
  case 'C':
    console.log("Average");
```

```
    break;
  default:
    console.log("Invalid grade");
}
```

Output:

Good

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

Theory Assignment

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

basic example of each.

In JavaScript, **loops** are used to repeat a block of code multiple times until a certain condition is met.
The main types of loops are **for**, **while**, and **do-while** loops.

1. for Loop

Used when you know **how many times** you want to repeat the code.

Syntax:

```
for (initialization; condition; increment/decrement) {
  // 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 you want to repeat the code **as long as** a condition is true.

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

This loop runs the code **at least once**, even if the condition is false.

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
```

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

Feature	while loop	do-while loop
Condition Check	Condition is checked **before** running the loop.	Condition is checked **after** running the loop.
Execution	The loop runs **only if** the condition is true.	The loop runs **at least once**, even if the condition is false.

Syntax Example	javascript
	while(condition) {
	 // code
	}
		javascript
		do {
		 // code
		} while(condition);

6. Functions

Theory Assignment

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

In JavaScript, a **function** is a block of code designed to perform a specific task.
It helps to **reuse code** — you can define a function once and use it many times.

Syntax for Declaring a Function:

```
function functionName(parameters) {
  // code to be executed
}
```

Syntax for Calling a Function:

```
functionName(arguments);
```

Example:

```
function greet(name) {        // Function declaration
  console.log("Hello, " + name + "!");
}

greet("Zeel");                // Function call
```

Output:

Hello, Zeel!

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

Feature	Function Declaration	Function Expression
Syntax	Declared using the function keyword with a name.	A function assigned to a variable (can be anonymous).
Hoisting	Hoisted — can be called **before** it's defined.	Not hoisted — can be called **only after** it's defined.

Example	javascript
	function greet() {
	 console.log("Hello!");
	}
		javascript
		const greet = function() {
		 console.log("Hello!");
		};

Calling	greet(); (works before or after declaration)	greet(); (works only after declaration)

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

1. Parameters:

- Parameters are **placeholders** for values that are passed **into a function**.
- They allow a function to work with different inputs each time it is called.

Example:

```
function add(a, b) {   // a and b are parameters
  console.log(a + b);
}
add(5, 3);             // 5 and 3 are arguments
```

Output:

8

2. Return Values:

- The **return** statement is used to **send a value back** from the function to where it was called.
- Once a return is executed, the function **stops running**.

Example:

```
function multiply(x, y) {
  return x * y;        // returns the result
}
let result = multiply(4, 5);
console.log(result);
```

Output:

20

7. Arrays

Theory Assignment

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

An **array** in JavaScript is a **collection of multiple values** stored in a single variable. It allows you to store and manage a list of items such as numbers, strings, or objects.

Declaring an Array:

You can declare an array in two ways:

1. **Using square brackets []:**

```
let fruits = ["apple", "banana", "mango"];
```

2. **Using the new Array() constructor:**

```
let numbers = new Array(10, 20, 30, 40);
```

Accessing Array Elements:

Array elements are accessed using **index numbers**, starting from **0**.

```
console.log(fruits[0]); // Output: apple
console.log(fruits[2]); // Output: mango
```

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

1. push()

- Adds one or more elements **to the end** of the array.
- Returns the **new length** of the array.

Example:

```
let fruits = ["apple", "banana"];
fruits.push("mango");
console.log(fruits); // ["apple", "banana", "mango"]
```

2. pop()

- Removes the **last element** from the array.
- Returns the **removed element**.

Example:

```
let fruits = ["apple", "banana", "mango"];
fruits.pop();
console.log(fruits); // ["apple", "banana"]
```

3. shift()

- Removes the **first element** from the array.
- Returns the **removed element**.

Example:

```
let fruits = ["apple", "banana", "mango"];
fruits.shift();
console.log(fruits); // ["banana", "mango"]
```

4. unshift()

- Adds one or more elements **to the beginning** of the array.
- Returns the **new length** of the array.

Example:

```
let fruits = ["banana", "mango"];
fruits.unshift("apple");
console.log(fruits); // ["apple", "banana", "mango"]
```

Theory Assignment

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

An **object** in JavaScript is a **collection of key–value pairs** used to store related data and functions together.
Each key (called a **property**) has a value, which can be a number, string, array, or even another object.

Difference between Objects and Arrays:

Feature	Object	Array
Structure	Stores data as **key–value pairs**	Stores data as **ordered list of items**
Access	Accessed using **keys** (e.g., person.name)	Accessed using **index numbers** (e.g., fruits[0])
Use Case	Used for storing **structured data** (like details of a person, product, etc.)	Used for **lists or collections** of similar items
Order	Properties are **not ordered**	Elements are **ordered**

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

bracket notation.

1. Dot Notation (.)

- The most common and simple way to access or update properties.
- You directly use the property name after a dot (.).

Example:

```
let person = { name: "Zeel", age: 20 };

// Accessing property
console.log(person.name); // Output: Zeel

// Updating property
person.age = 21;
console.log(person.age); // Output: 21
```

2. Bracket Notation [ ]

- Property name is written as a **string inside square brackets**.
- Useful when the property name has **spaces, special characters, or is stored in a variable**.

Example:

```
let person = { name: "Zeel", age: 20, "home city": "Ahmedabad" };

// Accessing property
console.log(person["home city"]); // Output: Ahmedabad

// Updating property
person["age"] = 22;
console.log(person["age"]); // Output: 22
```

9. JavaScript Events

Theory Assignment

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

JavaScript **events** are actions or occurrences that happen in the browser and can be detected by JavaScript.
These events are triggered by user interactions or browser activities.

Examples of events:

- Clicking a button → onclick
- Moving the mouse → onmousemove
- Pressing a key → onkeydown
- Loading a web page → onload

Role of Event Listeners

An **event listener** is a function that **waits for a specific event to occur** and then runs a piece of code in response.
It helps separate JavaScript code from HTML for cleaner structure.

Question 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 an HTML element without overwriting existing events.
It allows you to listen for a specific event (like a click, keypress, or mouseover) and then run a function when that event happens.

```
<button id="myButton">Click Me</button>


<script>
  // Select the button element
  let btn = document.getElementById("myButton");
```

```
  // Add event listener for click
  btn.addEventListener("click", function() {
    alert("Button was clicked!");
  });
</script>
```

10. DOM Manipulation

Theory Assignment

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

The **DOM (Document Object Model)** is a **programming interface** that represents an HTML or XML document as a **tree structure**.
In this structure, **each element (tags, text, attributes, etc.)** is represented as an **object (node)** that JavaScript can access and manipulate.

In simple terms, the DOM allows JavaScript to **interact with and change** the content, structure, and style of a webpage **dynamically**.

How JavaScript Interacts with the DOM

JavaScript can:

1. **Access elements** on the web page.
2. **Change content or style** of elements.
3. **Add or remove elements** dynamically.
4. **Handle user events** (like clicks, key presses, etc.).

□ Question 2: Explain the methods getElementById(), getElementsByClassName(), and

querySelector() used to select elements from the DOM.

1. getElementById()

- Used to **select a single element** by its **ID**.
- It returns **only one element** because IDs are unique in a page.

Syntax:

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

Example:

```
<p id="demo">Hello!</p>

<script>
let element = document.getElementById("demo");
element.style.color = "blue";
</script>
```

2. getElementsByClassName()

- Used to **select multiple elements** that share the **same class name**.
- It returns a **collection (array-like list)** of all matching elements.

Syntax:

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

Example:

```
<p class="text">Hello</p>
<p class="text">World</p>

<script>
let items = document.getElementsByClassName("text");
items[0].style.color = "red";
items[1].style.color = "green";
</script>
```

3. querySelector()

- Used to **select the first element** that matches a **CSS selector** (like #id, .class, or tag).
- More flexible because it works like CSS.

Syntax:

```
document.querySelector("selector");
```

Example:

```
<p class="msg">Welcome!</p>

<script>
let el = document.querySelector(".msg");
el.style.fontSize = "20px";
</script>
```

11. JavaScript Timing Events (setTimeout, setInterval)

Theory Assignment

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

1. setTimeout()

- Used to **run a function once** after a specified amount of time (in milliseconds).
- It executes code **after a delay**.

Syntax:

```
setTimeout(function, delay);
```

Example:

```
setTimeout(function() {
  alert("Hello after 3 seconds!");
}, 3000); // 3000 milliseconds = 3 seconds
```

🕒 The message will appear **once after 3 seconds**.

2. setInterval()

- Used to **run a function repeatedly** at a given time interval (in milliseconds).
- It keeps executing the code until stopped using clearInterval().

Syntax:

```
setInterval(function, interval);
```

Example:

```
setInterval(function() {
  console.log("This runs every 2 seconds");
}, 2000);
```

The message will print **every 2 seconds continuously**.

Stopping the Timers

You can stop them using:

- clearTimeout(timerID) → stops a timeout.
- clearInterval(intervalID) → stops a repeating interval.

Example:

```
let timer = setInterval(function() {
  console.log("Running...");
}, 1000);

setTimeout(function() {
  clearInterval(timer);
```

```
  console.log("Stopped!");
}, 5000);
```

12. JavaScript Error Handling

Theory Assignment

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

1. try block

- The code that **might cause an error** is placed inside the try block.

2. catch block

- If an error occurs in the try block, the catch block **handles the error** and prevents the program from crashing.

3. finally block

- The code inside the finally block **always runs**, whether an error occurs or not. (It's often used for cleanup tasks.)

Example:

```
try {
  let num = 10;
  console.log(num / 0);
  console.log(unknownVar);  // This will cause an error
}
catch (error) {
  console.log("An error occurred: " + error.message);
}
finally {
  console.log("Code inside finally always runs.");
}
```

Output:

Infinity
An error occurred: unknownVar is not defined
Code inside finally always runs.

Question 2: Why is error handling important in JavaScript applications?

Error handling is important in JavaScript because it helps make applications **more stable, secure, and user-friendly**.
Without it, even a small error could **stop the entire program** from running.

Reasons Why Error Handling Is Important:

1. **Prevents Program Crashes**
    a. Errors are caught and handled properly, so the whole application doesn't stop working suddenly.
2. **Gives Useful Error Messages**
    a. Instead of showing confusing technical errors, you can display clear messages to users (like "Something went wrong, please try again.").
3. **Helps Debugging**
    a. Makes it easier for developers to **find and fix** problems in the code.
4. **Maintains Smooth User Experience**
    a. Even if errors occur, the app continues to run normally without freezing or breaking.
5. **Improves Security**
    a. Proper error handling ensures that **sensitive information** (like system errors or file paths) isn't exposed to users.