❖ **JavaScript Introduction**

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

Ans :- JavaScript is a programming language used to create dynamic and interactive web pages.

Role in Web Development:

- Enhances user experience by enabling interactivity (e.g., animations, form validation).
- Works with HTML and CSS to build responsive and dynamic web pages.
- Enables client-side scripting for real-time updates without reloading the page.
- Used in frameworks (React, Angular, Vue) and backend (Node.js) for full-stack development.

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

Ans :- JavaScript vs. Python vs. Java

1. **Type System**
   - **JavaScript & Python**: Dynamically typed
   - **Java**: Statically typed
2. **Execution Environment**
   - **JavaScript**: Browser & Node.js
   - **Python**: Backend, data science, automation
   - **Java**: JVM, enterprise apps, Android
3. **Syntax**
   - **JavaScript & Java**: Use {} and ;
   - **Python**: Uses indentation
4. **Object Model**
   - **JavaScript**: Prototype-based
   - **Python & Java**: Class-based

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

Ans:- Use of <script> Tag in HTML

- The <script> tag is used to include JavaScript code in an HTML document.
- It can be placed inside the <head> or <body>.
- JavaScript inside <script> can manipulate HTML, CSS, and handle events.
- Place it before the closing </body> for better performance.
- No need to use <script> tags inside the external file.

- ❖ **Variables and Data Types :-**

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

Ans:-

1. **Var :-**
   - **var** (Old method)
   - Function-scoped.
   - Can be redeclared and reassigned.
2. **Let :-**
   - Block-scoped.
   - Can be reassigned but not redeclared.
3. **Const:-**
   - Block-scoped.
   - Cannot be reassigned or redeclared.

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

Ans :- There are two types of data types:-

1. **Primitive Type:-**
   - **Number**: Represents numeric values.
     Eg:- let num = 42;
   - **String**: Represents sequences of characters.
     Eg:- let name = "John";
   - **Boolean**: Represents true or false.
     Eg:- let isActive = true;
   - **Undefined**: A variable that has been declared but not assigned a value.
     Eg:- let a;
     console.log(a);
   - **Null**: Represents the intentional absence of any value.
     Eg:- let obj = null;
   - **Symbol** (ES6): A unique and immutable data type used for object properties.
     Eg:- let sym = Symbol('unique');
   - **BigInt** (ES11): Represents large integers beyond the Number type limit.
     Eg:- let bigNum = 9007199254740991n;

2. **Non-primitive type:-**
   Object: A collection of properties (key-value pairs)
   Eg:- let person = { name: "Alice", age: 30 };

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

Ans:- undefined: A variable is declared but not assigned a value.

Eg :- let x;

console.log(x);

null: A variable is explicitly assigned a "no value" state.

Eg:- let y = null;

console.log(y);

❖ JavaScript Operators :-

Question 1: What are the different types of operators in JavaScript? Explain with examples. Arithmetic operators, Assignment operators, Comparison operators, Logical operators.

Ans :- 1. Arithmetic Operators:- + , - , * , / , % , ++ , --.

Eg:- let a = 10, b = 5;

console.log(a + b); // 15

console.log(a % b); // 0

2. **Assignment Operators :-** = , += , -= , *= , /= , %=

```
Eg :- let x = 10;
        x += 5; // x = 15
        console.log(x);
```

3. **Comparison Operators :-** == , === , != , > , < , >= , <=

```
Eg:-   console.log(10 > 5);  // true
       console.log(5 == "5"); // true (type conversion)
       console.log(5 === "5"); // false (strict comparison)
```

4. Logical Operators :- &&, ` , !

```
Eg :-  let isAdult = true;
       let hasID = false;
```

```
console.log(isAdult && hasID); // false (both must be true)
console.log(isAdult || hasID); // true (at least one is true)
 console.log(!isAdult); // false (negation)
```

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

Ans :- 1. == (Loose Equality) :- Converts types before comparison.

Eg:- console.log(5 == "5"); // true (string "5" is converted to number 5)

console.log(true == 1); // true (true is converted to 1)

3. **=== (Strict Equality) :-** Compares both value and type.
   Eg:- console.log(5 === "5"); // false (number vs. string)
   console.log(true === 1); // false (boolean vs. number)

❖ **Control Flow :-**

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

Ans :- Control flow refers to the order in which statements are executed in a program. JavaScript executes code line by line, but control structures like conditional statements, loops, and functions alter this flow based on conditions or logic.

if-else Statement in JavaScript

The if-else statement allows decision-making in JavaScript based on conditions.

Eg :- let num = -5;

```
if (num > 0) {

   console.log("The number is positive.");

} else {

   console.log("The number is negative or zero.");

}
```

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

Ans:- The switch statement is used for **decision-making** when multiple possible values need to be checked against a single variable or expression. It provides a **cleaner** and **more readable** alternative to multiple if-else if conditions.

```
let day = 3;

switch (day) {

  case 1:

    console.log("Monday");

    break;

  case 2:

    console.log("Tuesday");

    break;

  case 3:

    console.log("Wednesday");

    break;

  case 4:

    console.log("Thursday");

    break;

  case 5:

    console.log("Friday");

    break;

  case 6:

  case 7:

    console.log("Weekend!");

    break;

  default:
    console.log("Invalid day");

}
```

Example: When to Use **if-else** Instead

If checking a **range** (not a fixed value), use if-else:

```
let age = 18;


if (age < 12) {

  console.log("Child");

} else if (age < 18) {

  console.log("Teen");

} else {

  console.log("Adult");

}
```

- ❖ Loops (For, While, Do-While) :-

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

Ans:- 1. for Loop

Used when we know **how many times** we want to execute the loop.

```
Eg:-  for (let i = 1; i <= 5; i++) {

        console.log(i);

      }
```

2. while Loop

Used when **the number of iterations is unknown**, and we repeat until a condition becomes false.

Eg:- let i = 1;

```
    while (i <= 5) {
         console.log(i);
         i++;
       }
```

3. do-while Loop

Similar to while, but **executes at least once**, even if the condition is false.

Eg:- let i = 1;

```
       do {
         console.log(i);
         i++;
       } while (i <= 5);
```

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

Ans:- while Loop :-

```
       let i = 5;
       while (i < 5) {
         console.log("This will not run");
       }
```

do-while Loop :-

```
       let i = 5;
       do {
         console.log("This will run at least once");
       } while (i < 5);
```

❖ Functions :-

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

Ans:- A function is a reusable block of code that performs a specific task. Functions help **in** code reusability, modularity, and better readability**.**

Eg:-
```
function addNumbers(a, b) {  // Function declaration

    return a + b;           // Returns the sum

}

   let sum = addNumbers(5, 3);  // Function call

   console.log(sum);           // Output: 8
```

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

Ans:-

 Declaration is hoisted, so it can be called before it's defined.

 Expression is not hoisted, so it must be defined before calling

Function Declaration (Hoisted) :-

```
greet(); //

function greet() {

   console.log("Hello");

}
```

Function Expression (Not Hoisted) :-

```
greet(); //  Error: Cannot access 'greet' before initialization

let greet = function() {

   console.log("Hello");

};
```

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

Ans :- 1. Parameters in Functions

Parameters are **variables** used to pass information into a function. They act as **placeholders** for values you provide when calling the function.

```
function add(a, b) {

   return a + b;

}
```

add(5, 3);

2. Return Values in Functions:-   A return value is the **output** the function gives back after performing its task. It can be used in other parts of the program.

```
function add(a, b) {

   return a + b;

}

let result = add(5, 3);

console.log(result); // Output: 8
```

❖ **Arrays :-**

Question 1: What is an array in JavaScript? How do you declare and initialize an array?
Ans:- An array in JavaScript is a special variable that can hold multiple values at once. Arrays are ordered lists that can store elements of any data type (numbers, strings, objects, etc.).

1. Declaring an Array :-

You declare an array using square brackets **[].**

Eg:- let arr = []; // Empty array.

2. Initializing an Array :-

You can initialize an array with values inside the square brackets, separated by commas.

Eg:- let fruits = ["Apple", "Banana", "Cherry"];

       let numbers = [1, 2, 3, 4, 5];

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

Ans:- **1. push() Method :-**

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

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

fruits.push("Cherry", "Date");  // Adds 'Cherry' and 'Date' at the end

console.log(fruits);  // Output: ["Apple", "Banana", "Cherry", "Date"]
```

2. pop() Method :-

- **Purpose:** Removes the **last** element from an array.
- **Returns:** The removed element.

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

let removedFruit = fruits.pop();  // Removes 'Cherry'

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

console.log(removedFruit);  // Output: "Cherry"
```

3. shift() Method :-

- **Purpose:** Removes the **first** element from an array.
- **Returns:** The removed element.

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

let removedFruit = fruits.shift();  // Removes 'Apple'

console.log(fruits);  // Output: ["Banana", "Cherry"]

console.log(removedFruit);  // Output: "Apple"
```

4. unshift() Method :-

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

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

fruits.unshift("Apple", "Mango");  // Adds 'Apple' and 'Mango' at the beginning

console.log(fruits);  // Output: ["Apple", "Mango", "Banana", "Cherry"]
```

❖ **Objects:-**

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

Ans:- Object: A collection of **key-value pairs**. Keys are strings (or numbers), and values can be any data type.

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

Array: An ordered list of elements indexed by numbers.

Eg:- let fruits = ["Apple", "Banana"];

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

Ans:- **Dot Notation**:

- **Access**: object.property
- **Update**: object.property = value

Eg:- person.name = "Bob";

Bracket Notation:

- **Access**: object["property"]
- **Update**: object["property"] = value

Eg:- person["age"] = 30;

❖ **JavaScript Events :-**

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

Ans:- JavaScript events are **actions or occurrences** that happen in the browser, such as a user clicking a button, submitting a form, or resizing a window.

Adding an Event Listener: Use addEventListener() to bind a function to an event.

Eg:- button.addEventListener("click", function() {

```
        alert("Button clicked!");

    });
```

Common Events: click, mouseover, keydown, submit, etc.

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

Ans:- **addEventListener()**

The addEventListener() method is used to **attach an event handler** to a specific event on an element. It listens for a particular event and executes a callback function when the event occurs.

Eg:- element.addEventListener(event, function, useCapture);

- **event**: The type of event (e.g., click, mouseover).
- **function**: The function to be executed when the event is triggered.
- **useCapture**: Optional. Determines if the event should be captured during the capturing phase (default is false).

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

     button.addEventListener("click", function() {

        alert("Button clicked!");

     });
```

Explanation: When the button with the ID myButton is clicked, the **alert** will pop up.

❖ **DOM Manupulation :-**

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

Ans:- The **DOM** is a programming interface for web documents. It represents the document as a tree structure where each node is an object representing a part of the page, such as elements, attributes, and text.

How JavaScript Interacts with the DOM:

JavaScript can **access, modify, and manipulate** the DOM using methods and properties like:

- **Accessing elements**: document.getElementById(), document.querySelector()
- **Modifying content**: element.innerHTML, element.textContent
- **Changing styles**: element.style.property
- **Adding/removing elements**: element.appendChild(), element.removeChild()

let para = document.getElementById("myPara");

para.textContent = "Hello, World!"; // Changes text content of the paragraph.

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

1. getElementById()

- Purpose: Selects a single element by its ID.
- Returns: The first element with the matching ID (or null if not found).

  Eg:- let element = document.getElementById("myElement");

2. getElementsByClassName()

- **Purpose**: Selects all elements with the specified **class name**.
- **Returns**: A **HTMLCollection** of elements.

  Eg:- let elements = document.getElementsByClassName("myClass");

3. querySelector()

- **Purpose**: Selects the **first element** that matches a **CSS selector**.
- **Returns**: The first matching element (or null if not found).

  Eg:- let element = document.querySelector(".myClass"); // Selects the first element with class "myClass".

❖ **JavaScript Timing Events (setTimeout, setInterval):-**

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

Ans:- 1. **setTimeout()**

1.**Purpose**: Executes a function after a specified delay (in milliseconds).
Eg:- setTimeout(function, delay);

2.Use: Runs the function once after the given time interval.

```
Eg:-   setTimeout(function() {

   alert("This message appears after 3 seconds");
}, 3000);  // 3000 milliseconds = 3 seconds
```

2. **setInterval()**

1.Purpose: Executes a function at **regular intervals** (in milliseconds).

```
Eg:-   setInterval(function, interval);
```

2. Use: Runs the function repeatedly after every specified interval.

```
Eg:-  setInterval(function() {
          console.log("This message repeats every 2 seconds");
}, 2000);  // 2000 milliseconds = 2 seconds
```

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

Ans:- **Eg:-** setTimeout(function() {

```
          console.log("This message appears after 2 seconds");
    }, 2000);  // 2000 milliseconds = 2 seconds.
```

❖ **JavaScript Error Handling:-**

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

Ans:- Error handling in JavaScript allows you to **manage errors** in your code without stopping the entire program. The try, catch, and finally blocks are used to handle exceptions.

1. **try** Block

- **Purpose**: Contains code that might throw an error.

2. catch Block

- **Purpose**: Catches and handles the error if one occurs in the try block.

3. finally Block

- **Purpose**: Executes code **after** the try and catch blocks, regardless of whether an error occurred or not.

```
try {

   let result = 10 / 0;  // This will cause Infinity, not an error

   console.log(result);

} catch (error) {

   console.log("An error occurred:", error.message);  // Catching any error

} finally {

   console.log("This will run regardless of error");  // Always runs

}
```

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

Ans:-

1. **Prevents Application Crashes**:
   - Proper error handling ensures that an error doesn't crash the entire application. Without it, uncaught errors can stop your application unexpectedly.
2. **Improves User Experience**:
   - Instead of a blank page or a broken feature, users see informative messages, helping them understand what's wrong without disrupting their experience.
3. **Debugging**:
   - Error handling allows developers to **catch and log** errors, making it easier to identify and fix issues during development or in production.
4. **Graceful Degradation**:
   - When an error occurs, the application can continue running, or fallback behavior can be triggered, preventing critical failures.
5. **Security**:

- Proper handling prevents sensitive error details from being exposed to end-users, which could potentially be exploited.
