| USN | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Internal Assessment Test 1 – Set 1 –March 2026 | | | | | | | | |
| Sub: | Full Stack Development | | | | Sub Code: | BIS601 | Branch | ISE |
| Date: | 2/3/26 | Duration: | 90 min's | Max Marks: 50 | Sem/Sec: | VI / A, B & C | | OBE |

# Solution

| | | MARKS | CO | RBT |
|---|---|---|---|---|
| 1 a) | Explain the difference between var, let and const with suitable examples. | 5 | CO1 | L2 |
| | var, let, and const are used to declare variables in JavaScript. They differ in **scope, redeclaration, reassignment, and hoisting behavior**. | 2*2+1 | 5M | |

**var**
**Features:**

- **Function scoped**
- Can be **redeclared**
- Can be **reassigned**
- Hoisted and initialized with undefined
- No block scope

**Example:**

```
function test() {
  var x = 10;
  if (true) {
    var x = 20;  // Same variable (overwrites)
    console.log(x); // 20
  }
  console.log(x); // 20
}
test();
```

**Hoisting Example:**

```
console.log(a); // undefined
var a = 5;
```

**let**
**Features:**

- **Block scoped** (limited to { })
- Cannot be redeclared in the same scope
- Can be reassigned
- Hoisted but not initialized (Temporal Dead Zone)

**Example:**

```
function test() {
  let x = 10;
  if (true) {
    let x = 20;  // Different variable (block scope)
    console.log(x); // 20
  }
  console.log(x); // 10
}
test();
```

**Redeclaration Error:**

```
let a = 5;
let a = 10; // Error
```

**Temporal Dead Zone:**

```
console.log(b); // ReferenceError
let b = 5;
```

**const**
**Features:**

- **Block scoped**
- Cannot be redeclared
- Cannot be reassigned
- Must be initialized at declaration
- Hoisted but in Temporal Dead Zone **Example:**

| | | | | |
|---|---|---|---|---|
| | const x = 10;<br>console.log(x); // 10<br>**Reassignment Not Allowed:**<br>const x = 10;<br>x = 20; // Error | | | |
| 1 b) | Explain the structure of a JavaScript function. How are parameters and return values used? | 5 | CO1 | L2 |
| | **Structure of a JavaScript Function**<br>A **function** in JavaScript is a reusable block of code designed to perform a specific task. | | 5 | |
| | **Basic Structure (Syntax)**<br>function functionName(parameter1, parameter2) {<br>   // Code to be executed<br>   return value;   // optional<br>}<br>**Parts of a Function:**<br>1. **function keyword** → Used to declare a function<br>2. **functionName** → Name of the function<br>3. **Parameters** → Inputs given to the function<br>4. **Function body { }** → Code block that executes<br>5. **return statement** → Sends result back to caller (optional) | 2 | | |
| | **Example of a Simple Function**<br>function greet() {<br>   console.log("Hello World");<br>}<br>greet();<br>Here:<br>• No parameters<br>• No return value<br>• Just executes code | 2 | | |
| | **Parameters in JavaScript**<br>**What are Parameters?**<br>Parameters are variables listed inside parentheses in function definition.<br>They act as placeholders for values passed to the function. | 1 | | |
| | **Example with Parameters**<br>function add(a, b) {<br>   console.log(a + b);<br>}<br>add(5, 3);<br>**Explanation:**<br>• a and b → Parameters<br>• 5 and 3 → Arguments (actual values passed)<br>• Output → 8 | | | |
| | **Parameter Flow:**<br>Arguments → Parameters → Function Body → Output | | | |
| | **Types of Parameters**<br>**Default Parameters**<br>function greet(name = "Guest") {<br>   console.log("Hello " + name);<br>}<br>greet();        // Hello Guest<br>greet("Shilpa"); // Hello Shilpa | | | |

| | | | | |
|---|---|---|---|---|
| 2 | Write a program that creates an array of 5 cities and performs the following: i) Adds a city at the end ii) Removes the first city iii) Logs the total number of cities iv) Finds the index of a special city v) Searches for a specific city vi) Replaces a specific city with another | 10 | CO1 | L3 |
| | import promptSync from 'prompt-sync';<br>//<br>const prompt = promptSync();<br><br>// Prompt the user to enter 5 city names, separated by commas<br>let input = prompt("Enter 5 cities separated by commas: ");<br>let cities = input.split(',').map(city => city.trim());<br>console.log("Initial cities:", cities);<br><br>// Log the total number of cities<br>console.log("Total number of cities:", cities.length);<br><br>// Add a new city at the end<br>let newCity = prompt("Enter a city to add to the end: ");<br>cities.push(newCity);<br>console.log("Cities after adding a new one:", cities);<br><br>// Remove the first city<br>console.log("Removing the first city:", cities[0]); // Log the first city before removal<br>cities.shift();<br>console.log("Cities after removing the first one:", cities);<br><br>// Find and log the index of a specific city<br>let searchCity = prompt("Enter a city to find its index: ");<br>let cityIndex = cities.indexOf(searchCity);<br>console.log("Index of", searchCity + ":", cityIndex !== -1 ? cityIndex : "City not found"); | 2<br>2<br>2<br>2<br>2 | 10 | |
| 3a) | Discuss the differences in how arrays are constructed and accessed and explain when to use an array vs. an object in various programming scenarios. | 5 | CO1 | L2 |
| | Arrays store ordered data accessed using index numbers.<br>Objects store data in key–value pairs and are used for structured information.<br><br>Array Example:<br>let arr = ["Math", "Science"];<br><br>Object Example:<br>let student = {<br>    name: "Rahul",<br>    grade: "A",<br>    subjects: ["Math", "Science"],<br>    displayInfo: function() {<br>        console.log("Name:", this.name);<br>        console.log("Grade:", this.grade);<br>        console.log("Subjects:", this.subjects.join(", "));<br>    }<br>};<br><br>student.displayInfo();<br><br>Use Arrays when order matters.<br>Use Objects when data has properties or attributes. | 2<br>2<br>1 | 5 | |

| | | | | |
|---|---|---|---|---|
| 3b | Explain any 5 functions used on arrays for manipulation. | 5 | CO2 | L2 |
| | **push() – Add element at the end**<br>• Adds one or more elements to the end of an array.<br>• Modifies the original array.<br>• Returns new length.<br>**Example:**<br>let numbers = [1, 2, 3];<br>numbers.push(4);<br><br>console.log(numbers); // [1, 2, 3, 4]<br>**pop() – Remove element from the end**<br>• Removes the last element.<br>• Modifies the original array.<br>• Returns the removed element.<br>**Example:**<br>let numbers = [1, 2, 3];<br>let removed = numbers.pop();<br><br>console.log(numbers); // [1, 2]<br>console.log(removed); // 3<br>**shift() – Remove element from the beginning**<br>• Removes the first element.<br>• Modifies the original array.<br>• Returns the removed element.<br>**Example:**<br>let numbers = [10, 20, 30];<br>numbers.shift();<br><br>console.log(numbers); // [20, 30]<br><br>**unshift() – Add element at the beginning**<br>• Adds one or more elements at the start.<br>• Modifies the original array.<br>• Returns new length.<br>**Example:**<br>let numbers = [20, 30];<br>numbers.unshift(10);<br><br>console.log(numbers); // [10, 20, 30]<br>**splice() – Add/Remove elements at specific position**<br>• Can add, remove, or replace elements.<br>• Modifies the original array.<br>**Syntax:**<br>array.splice(start, deleteCount, item1, item2...)<br>**Example (Remove element):**<br>let numbers = [1, 2, 3, 4];<br>numbers.splice(1, 2);<br><br>console.log(numbers); // [1, 4]<br>**Example (Add element):**<br>let numbers = [1, 4];<br>numbers.splice(1, 0, 2, 3);<br><br>console.log(numbers); // [1, 2, 3, 4] | 1M*5 | 5 | |
| 4 | Create a button in your HTML with the text **“Click Me”**. Add an event listener to log **“Button clicked!”** to the console when the button is clicked. Select an image and add a **mouseover** event listener to change | 10 | CO2 | L3 |

| | | | | |
|---|---|---|---|---|
| | its border color. Add an event listener to the document that logs the key pressed by the user. | | | |
| | `<!DOCTYPE html>`<br>`<head>`<br><br>`   <title>Event Listeners Example</title>`<br>`</head>`<br>`<body>`<br><br>`   <!-- Button to click -->`<br>`   <button id="clickButton">Click Me</button>`<br><br>`   <!-- Example image with a reliable URL -->`<br>`   <img id="exampleImage" src="https://www.w3schools.com/html/img_chania.jpg"`<br>`alt="Example Image" width="200">`<br><br>`   <script>`<br>`      // Event listener for the button click`<br>`      document.getElementById('clickButton').addEventListener('click', function() {`<br>`         console.log("Button clicked!");`<br>`      });`<br><br>`      // Event listener for mouseover on the image to change its border color`<br>`      document.getElementById('exampleImage').addEventListener('mouseover', function() {`<br>`         this.style.border = '5px solid red';  // Change border color to red`<br>`      });`<br><br>`      // Event listener to log the key pressed by the user`<br>`      document.addEventListener('keydown', function(event) {`<br>`         console.log("Key pressed: " + event.key);`<br>`      });`<br><br>`      // Event listener for mouseout on the image to change its border color`<br>`      document.getElementById('exampleImage').addEventListener('mouseout', function() {`<br>`         this.style.border = 'none';  // remove the border`<br>`      });`<br><br>`   </script>`<br><br>`</body>`<br>`</html>` | 2<br>2<br>2<br>2<br>2 | 10 | |
| 5a) | What are event listeners in JavaScript? How do they differ from traditional event attributes (like onclick) for binding events? | 5 | CO2 | L2 |
| | **What are Event Listeners in JavaScript?**<br>An **event listener** is a method used to attach an event handler function to an HTML element so that a specific action is performed when an event occurs.<br>Common events:<br>• click<br>• mouseover<br>• keydown<br>• submit<br>• change | 2 | 5 | |
| | **Syntax of Event Listener**<br>element.addEventListener("event", functionName);<br>or<br>element.addEventListener("event", function() { | | | |

```
    // code to execute
});
```

**Example of Event Listener**

2

```
<button id="btn">Click Me</button>
let button = document.getElementById("btn");

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

When the button is clicked, the function executes.

**What are Traditional Event Attributes?**

1

Traditional event handling uses HTML attributes like:

- onclick
- onmouseover
- onsubmit

**Example:**

```
<button onclick="showMessage()">Click Me</button>

<script>
function showMessage() {
    alert("Button Clicked!");
}
</script>
```

Here, the event is directly written inside the HTML tag.

| | | | | |
|---|---|---|---|---|
| 5b | Explain any 5 different DOM methods used to access or manipulate HTML elements in JavaScript, including their syntax, use cases, and when each is preferred. | 10 | CO2 | L2 |

**Five DOM Methods to Access or Manipulate HTML Elements**

1M*5 5

The **DOM (Document Object Model)** allows JavaScript to access and modify HTML elements dynamically.

**getElementById()**

**Purpose:**

Selects an element by its **unique id**.

**Syntax:**

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

**Example:**

```
<p id="demo">Hello</p>
let element = document.getElementById("demo");
element.innerHTML = "Welcome";
```

**Use Case:**

- When element has a unique ID.
- Fastest and most direct selection.

**Preferred When:**

You need to access a single unique element.

**getElementsByClassName()**

**Purpose:**

Selects elements with a specific class name.

**Syntax:**

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

**Example:**

```
<p class="text">A</p>
<p class="text">B</p>
let elements = document.getElementsByClassName("text");
elements[0].style.color = "red";
```

**Use Case:**

- When multiple elements share the same class.

**⚠ Note:**
Returns an **HTMLCollection** (array-like object).
**Preferred When:**
You want to target elements grouped by class.

**getElementsByTagName()**
**Purpose:**
Selects elements based on tag name.
**Syntax:**
```
document.getElementsByTagName("tagName");
```
**Example:**
```
let paragraphs = document.getElementsByTagName("p");
paragraphs[0].style.fontWeight = "bold";
```
**Use Case:**

- When you want to access all elements of a particular type (like all <p> or <div>).

**Preferred When:**
You need bulk operations on specific tag types.

**querySelector()**
**Purpose:**
Selects the **first element** that matches a CSS selector.
**Sntax:**
```
document.querySelector("CSS selector");
```
**Example:**
```
let element = document.querySelector(".text");
element.style.backgroundColor = "yellow";
```
**Use Case:**

- Supports ID (#id)
- Class (.class)
- Tag (p)
- Complex selectors (div > p)

**Preferred When:**
You need flexible CSS-style selection.

**querySelectorAll()**
**Purpose:**
Selects **all elements** matching a CSS selector.
**Syntax:**
```
document.querySelectorAll("CSS selector");
```
**Example:**
```
let elements = document.querySelectorAll(".text");

elements.forEach(function(el) {
   el.style.color = "blue";
});
```

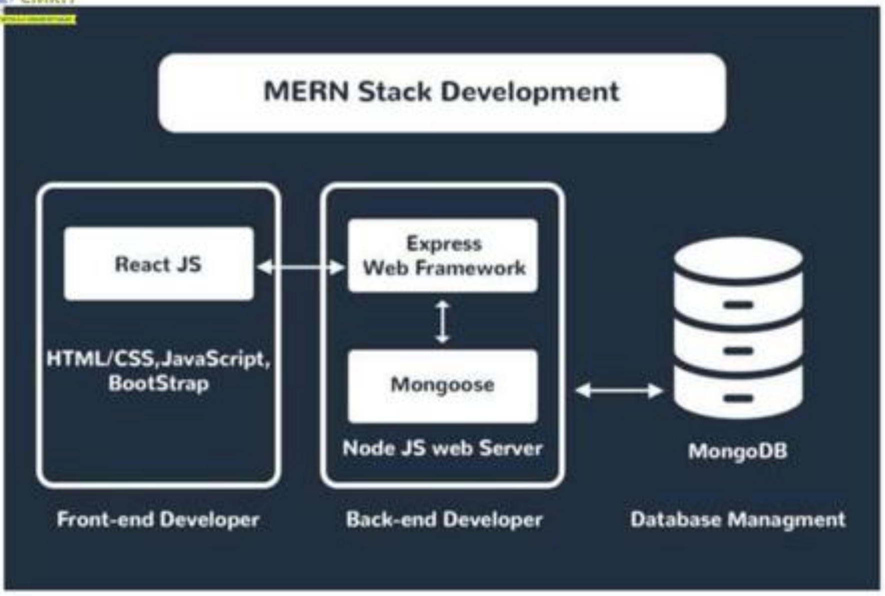
| | | | | |
|---|---|---|---|---|
| 6a | Explain and implement serverless Hello World Program. Explain each step. | 5 | CO3 | L2 |
| | | 4 | 5 | |

```
<!DOCTYPE HTML>
<html>

<head>
   <meta charset="utf-8">
   <title>Pro MERN Stack</title>

   <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
   <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
```

| | | | | |
|---|---|---|---|---|
| | `</head>`<br><br>`<body>`<br><br>`  <div id="contents"></div>`<br><br>`  <script>`<br>`    const element = React.createElement(`<br>`      'div',`<br>`      { title: 'Outer div' },`<br>`      React.createElement('h1', null, 'Hello World!')`<br>`    );`<br><br>`    ReactDOM.render(element, document.getElementById('contents'));`<br>`  </script>`<br><br>`</body>`<br><br>`</html>`<br><br>Explanation | 1 | | |
| 6b | Explain the components of the MERN stack and discuss how they interact in a full stack application. Highlight the role of each component with examples. | | | |
| | **1. Components of the MERN Stack and Their Interaction**<br>The MERN stack is a popular JavaScript-based technology stack used for developing full stack web applications.<br>MERN stands for MongoDB, Express.js, React.js, and Node.js. Each component plays a specific role in handling<br>different layers of the application.<br><br>MongoDB is a NoSQL database that stores application data in JSON-like documents. It is flexible and schema-less,<br>allowing developers to store structured as well as semi-structured data. For example, user information, product<br>details, and application logs can be stored in MongoDB collections.<br><br>Express.js is a lightweight backend web framework that runs on Node.js. It simplifies server-side programming by<br>providing routing, middleware support, and request handling. Developers use Express to create REST APIs such as<br>login APIs, product APIs, and data submission endpoints.<br><br>React.js is a front-end JavaScript library used to build interactive user interfaces. It follows a component-based<br>architecture where the UI is divided into reusable components. React manages the view layer and communicates with<br>the backend using HTTP requests (GET, POST, PUT, DELETE).<br><br>Node.js is the runtime environment that allows JavaScript to run on the server. It handles server operations,<br>processes client requests, and connects the frontend with the database.<br><br>Interaction Flow:<br>1. The user interacts with the React frontend.<br>2. React sends an API request to the Express/Node server.<br>3. The server processes the request and communicates with MongoDB.<br>4. MongoDB returns data to the server.<br>5. The server sends the response back to React, which updates the UI. | 1M*4 | 5 | |

| | | | | |
|---|---|---|---|---|
| |   MERN Stack Development<br>React JS<br>HTML/CSS,JavaScript, BootStrap<br>Express Web Framework<br>Mongoose<br>Node JS web Server<br>MongoDB<br>Front-end Developer<br>Back-end Developer<br>Database Managment<br><br>Example: In an online shopping application, React displays products, Express handles order requests, Node runs the server logic, and MongoDB stores user orders and product details. | 1 | | |