Mod 2

Q.1) What is event handling in Javascript? Explain it with an example.

⇒ ① Event handling in javascript refers to the process of responding to user interactions or actions, such as clicks, mouse movements, key presses or form submissions on a webpage.

② Javascript allows developers to define specific functions, called event handlers, that execute when an event occurs.

③ Working:

(i) Event: An action triggered by the user on the browser.

(ii) Event-listener: A mechanism in javascript to 'listen' for a specific event on an element.

(iii) Event handler: A function that gets executed when an event occurs.

④ Example:

```
<html>
<head>
</head>
<body>
    <h1 id="greeting"> Hello! </h1>
    <button id="changeText"> Change</button>
    <script
const button = document.getElementById('changeText');
button.addEventListener('click', () => {
document.getElementById('greeting').textContent = 'Hello, world!
  });
</script
```

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

(i) Here, the h1 is used to display a message 'Hello!'

(ii) A <button> element is used to trigger an action.

(iii) The getElementById method selects the button element by Id.

(iv) The addEventListener method is used to listen for the click event.

(v) when the button is clicked, the event handler changes the text of the <h1> element to "Hello, World!".

(5) other events :

(a) Mouseover event - (mouseover)

```
<html>
<head>
</head>
<body>
<h1 id='greeting'> hi </h1>
<script>
const heading = document.getElementById('greeting');
heading.addEventListener('mouseover', function (){
    heading.style.color = 'blue';
  });
</script>
</body>
</html>
```

(i) When the mouse hovers over the text 'hi', it turns to blue.

(b) Keypress event - (keypress)

(c) Form submission - (submit)

Page No.
Date

Q2) Explain exception handling in javascript with a suitable example.

⇒ ① Exception handling is a mechanism to handle runtime errors preventing the program from crashing.
② It uses try --- catch statement to manage errors and ensure the program continues to execute.
③ Components of exception handling
(i) try block:
(a) It contains the code that might throw an exception.
(b) If an exception occurs, the try block stops executing, and the control passes to the catch block.
(ii) catch block:
(a) Executes if an exception is thrown in the try block.
(b) It allows you to define how the error should be handled.
(iii) finally block:
(a) It executes after the try and catch blocks, regardless of whether an exception has occured or not.
(iv) throw Statement:
(a) Manually generates an exception.
(b) It can be used to throw custom errors.
④ Example:

```
function divideNumbers (a, b) {
  try {
  if (b==0) {
```

Page No.
Date

```
throw new Error ("Division by zero is not
                                   allowed");
}
const result = a/b;
console.log (`Result: ${result}`);
}
catch (error) {
console.error (`Error: ${error.message}`);
}
finally {
console.log ("Execution completed.");
}
}

divideNumbers (10, 2)     // Valid division
divideNumbers (10, 0)     // Triggers an exception
```

(i) when called the valid inputs (10, 2) the result is calculated and displayed.

(ii) when called with invalid divisor (10, 0), the custom error is caught and the error message is logged.

(iii) Output:

(1) Valid division:

```
Result: 5
Execution completed
```

(2) Division by zero:

```
Error: Division by zero is not allowed.
Execution completed.
```

Q3. Explain built in objects in Javascript.

=>

a. JavaScript's built-in objects are powerful tools that enable developers to handle a wide range of tasks with minimal effort.
b. By leveraging objects like String, Array, Date, and Math, developers can write cleaner, more efficient code.

c. Built-in Objects

1. **String**:
   a. Used to work with text data.
   b. Example:

```
let greeting = "Hello, World!";
console.log(greeting.toUpperCase());  // "HELLO, WORLD!"
```

2. **Number**:
   a. Represents numerical values.
   b. Example:

```
let num = 42;
console.log(num.toFixed(2));  // "42.00"
```

3. **Array**:
   a. Represents a collection of elements.
   b. Example:

```
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits.length);  // 3
fruits.push("Date");
console.log(fruits);  // ["Apple", "Banana", "Cherry", "Date"]
```

4. **Date**:
   a. Used to work with dates and times.
   b. Example:

```
let today = new Date();
console.log(today.toDateString());  // e.g., "Wed Nov 20 2024"
```

5. **Math**:

   a. Provides mathematical operations and constants.
   b. Example:

```
console.log(Math.PI);  // 3.141592653589793
console.log(Math.sqrt(16));  // 4
```

6. **JSON**:
   a. Used to parse and stringify JSON data.
   b. Example:

```
let jsonData = '{"name": "Alice", "age": 25}';
let obj = JSON.parse(jsonData);  // Convert JSON string to object
console.log(obj.name);  // "Alice"
let jsonString = JSON.stringify(obj); // Convert object to JSON string
console.log(jsonString);  // '{"name":"Alice","age":25}'
```

7. **Map and Set**:
   a. Map: Stores key-value pairs.
   b. Set: Stores unique values.
   c. Example:

```
let map = new Map();
map.set("key1", "value1");
console.log(map.get("key1"));  // "value1"

let set = new Set([1, 2, 3, 3]);
console.log(set.size);  // 3
```

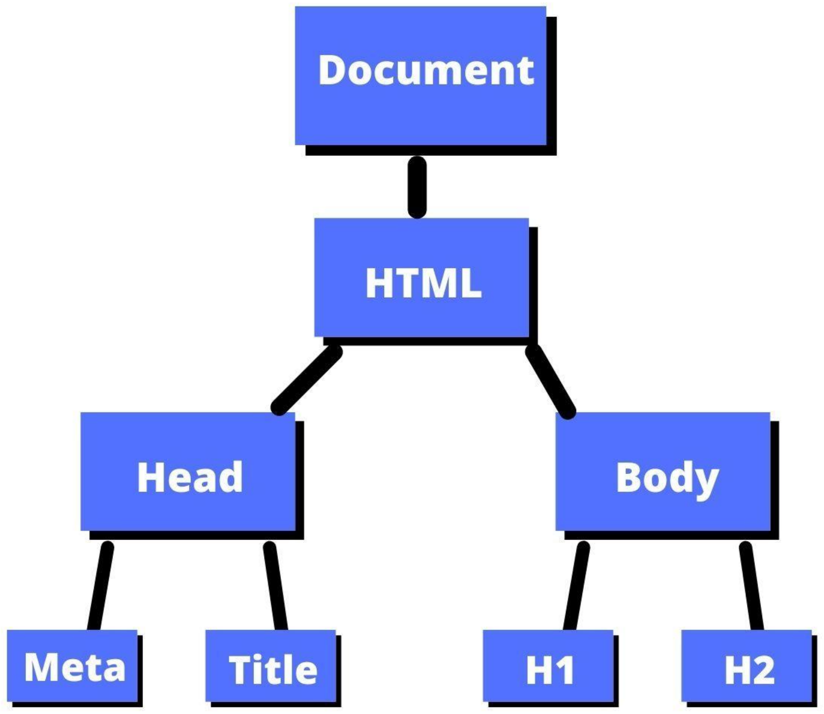
8. **Error Objects**:
   a. Handle runtime errors.
   b. Example:

```
try {
  throw new Error("An error occurred!");
} catch (error) {
  console.log(error.message);  // "An error occurred!"
}
```

Q4. Explain Javascript DOM Model

=>

a. The **Document Object Model (DOM)** is a programming interface for web documents.
b. It represents the structure of an HTML or XML document as a tree of objects that can be manipulated using JavaScript.
c. The DOM allows developers to dynamically access, modify, and update the content, structure, and style of a web page.



d. From the above diagram :

1. **Document** (Root node)
   Contains the HTML node as its child.
2. **HTML** (Parent node of Head and Body)
   Head (Parent node of `Meta` and `Title`)
   Body (Parent node of H1 and H2

e. DOM Access and Manipulation Examples

***1.* Accessing Nodes**

- **Accessing the title element**:

```
let titleElement = document.querySelector("title");
console.log(titleElement.innerText); // "Example Page"
```

- **Accessing the h1 element**:

```
let h1Element = document.querySelector("h1");
console.log(h1Element.innerText); // "Welcome to the DOM"
```

***2.* Modifying Elements**

- **Changing the content of the h2 element**:

```
let h2Element = document.querySelector("h2");
h2Element.innerText = "Exploring the DOM Tree";
```

- **Changing the title text dynamically**:

```
document.title = "New Title Example";
```

***3.* Adding New Element**

- **Adding a new paragraph to the body**:

```
let newParagraph = document.createElement("p");
newParagraph.innerText = "This is a dynamically added paragraph.";
document.body.appendChild(newParagraph);
```

4. Styling Elements

- **Styling the h1 element**:

```
let h1Element = document.querySelector("h1");
h1Element.style.color = "blue";
h1Element.style.fontSize = "32px";
```

***5.* Removing an Element**

- **Removing the meta tag**:

```
let metaElement = document.querySelector("meta");
metaElement.remove();
```