Advance JavaScript

Q.1; Write a program to Show an alert

Ans;
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Show Alert</title>
</head>
<body>



    <script>
        // Display an alert when the page loads
        window.onload = function() {
          alert("Hello! This is an alert.");
        };
      </script>
</body>
</html>
```

Q.2; What will be the result for these expressions?

1. 5 > 4

Ans; The expression 5>4 is a Boolean experience that evaluates to ‘true’.

2. . "apple" > "pineapple"

Ans;” apple” > “pineapple” evaluate ‘false ‘

3. "2" > "12"

Ans;’true”

4. undefined == null

Ans; true

5. undefined === null

Ans; undefined === null evaluates to false because they are of different types.

6. null == "\n0\n"

the case of null == "\n0\n", the null value only equals null or undefined using loose equality. Any other comparison with null results in false. Therefore, null will not equal the string "\n0\n".

the expression null == "\n0\n" will evaluate to false.

7. . null === +"\n0\n"

Ans; null === +"\n0\n" compares an object (null) with a number (0), and they are of different types. Therefore, the strict equality comparison will evaluate to false.

Q.3; Will alert be shown? if ("0") { alert('Hello'); }

Ans; Therefore, in the expression if ("0"), the condition is true, and the alert('Hello') statement inside the if block will execute, showing the alert with the message "Hello".

Q.4; What is the code below going to output?

alert(null || 2 || undefined)

Ans; null is falsy.

2 is truthy.

Since null is falsy, the || operator moves to the next operand, which is 2. Since 2 is a truthy value, the expression short-circuits at this point, and 2 will be the result of the entire expression.

Therefore, alert(null || 2 || undefined) will output 2 in the alert dialog

Q.5; The following function returns true if the parameter age is greater than 18. Otherwise, it asks for a confirmation and returns its result: function checkAge(age) { else { } } if (age> 18) { return true; } // ...return confirm (‘did parents allow you?');

Ans. This function checkAge takes an age parameter. If the age is greater than 18, it returns true. Otherwise, it prompts the user with a confirmation dialog asking, "Did parents allow you?" using the confirm function. The function then returns the result of the confirmation dialog, which will be true if the user confirms and false if they cancel.

Q.6; Replace Function Expressions with arrow functions in the code below: Function ask(question, yes, no) { if (confirm(question))yes(); else no(); } ask("Do you agree?", function() { alert("You agreed."); }, function() { alert("You canceled the execution."); } }

Ans;
```
const ask = (question, yes, no) => {
            if (confirm(question)) {
                yes();
            } else {
                no();
            }
        };

        ask(
            "Do you agree?",
            () => {
                alert("You agreed.");
            },
            () => {
                alert("You canceled the execution.");
            }
        );
```

The ask function is now written using arrow function syntax, and the function() expressions inside ask have been replaced with arrow functions () => {}. This makes the code more concise while maintaining the same functionality.

Data Types and Objects

Q.1; Write the code, one line for each action:

A; Create an empty object user

```
Ans; const user = {}; // This creates an empty object named user
```

B; Add the property name with the value John

```
Ans; const user = {};

        user.name = "John";
```

C; Add the property surname with the value Smith.

```
Ans; const user = { name: "John" };

user.surname = "Smith";
```

D; Change the value of the name to Pete.

```
Ans; const user = { name: "John", surname: "Smith" };

user.name = "Pete";
```

E; Remove the property name from the object.

```
Ans; const user = { name: "Pete", surname: "Smith" };

delete user.name;
```

Q.2; Is array copied?

```
let fruits = ["Apples", "Pear", "Orange"];

// push a new value into the "copy" let shoppingCart = fruits;
shoppingCart.push("Banana"); // what's in fruits?
```

alert(fruits.length); // ?

```
Ans;   <script>

let fruits = ["Apples", "Pear", "Orange"];
let shoppingCart = fruits;
shoppingCart.push("Banana");
alert(fruits.length);

    </script>
```

The fruits array and the shoppingCart array reference the same array in memory. Therefore, when you push "Banana" into shoppingCart, the fruits array is also affected because they're the same array.

As a result, the fruits.length will be 4, because the "Banana" item pushed into shoppingCart (which refers to the same array as fruits) increases the length of the array fruits as well.

Q.3; Map to names

let john = { name: "John", age: 25 };

let pete = { name: "Pete", age: 30 };

let mary = { name: "Mary", age: 28 };

let users = [john, pete, mary];

let names = /* ... your code */ alert(names);

// John, Pete, Mary

```
Ans;   <script>

let john = { name: "John", age: 25 };
let pete = { name: "Pete", age: 30 };
let mary = { name: "Mary", age: 28 };
```

```
let users = [john, pete, mary];

let names = users.map(user => user.name);
alert(names.join(", "));


    </script>
```

Q.4; Map to objects

let john = { name: "John", surname: "Smith", id: 1 }

; let pete = { name: "Pete", surname: "Hunt", id: 2 };

let mary = { name: "Mary", surname: "Key", id: 3 }

; let users = [john, pete, mary];

let usersMapped = /* ... your code ... */ /* usersMapped = [{ fullName: "John Smith", id: 1 }, { fullName: "Pete Hunt", id: 2 }, { fullName: "Mary Key", id: 3 }]

*/ alert(usersMapped[0].id) // 1 alert(usersMapped[0].fullName)

// John Smith

Ans; <script>

```
let john = { name: "John", surname: "Smith", id: 1 };
let pete = { name: "Pete", surname: "Hunt", id: 2 };
let mary = { name: "Mary", surname: "Key", id: 3 };

let users = [john, pete, mary];

let usersMapped = users.map(user => ({
  fullName: `${user.name} ${user.surname}`,
  id: user.id
}));

alert(usersMapped[0].id); // 1
```

```
alert(usersMapped[0].fullName); // John Smith



    </script>
```

Q.5; Sum the properties There is a salaries object with arbitrary number of salaries. Write the function sumSalaries(salaries) that returns the sum of all salaries using Object.values and the for..of loop.If salaries is empty, then the result must be 0.

let salaries = {

"John": 100,

"Pete": 300,

"Mary": 250 };

alert(sumSalaries(salaries)); // 650

Ans;

```
<script>

function sumSalaries(salaries) {
  let sum = 0;
  for (let salary of Object.values(salaries)) {
    sum += salary;
  }
  return sum;
}

let salaries = {
  "John": 100,
  "Pete": 300,
  "Mary": 250
};

alert(sumSalaries(salaries)); // Output: 650
```

```
    </script>
```

Q.6; Destructuring assignment We have an object: Write the Destructuring assignment that reads:

A;) Name property into the variable name.

Ans;

```
    <script>

let user = { name: "John", age: 30 };

let { name } = user;

console.log(name); // Output: "John"


    </script>
```

B. Year's property into the variable age.

Ans;

```
    <script>

let user = { name: "John", age: 30 };

let { age } = user;

console.log(age); // Output: 30


    </script>
```

C; isAdmin property into the variable isAdmin (false, if no such property)

Ans;

```
<script>
```

```
let user = { name: "John", age: 30 };

let { isAdmin = false } = user;

console.log(isAdmin); // Output: false



    </script>
```

D. let user = { name: "John", years: 30};

```
Ans; <script>

        let user = { name: "John", years: 30 };

        let { name, years } = user;

        console.log(name); // Output: "John"
        console.log(years); // Output: 30

    </script>
```

Q.7; Turn the object into JSON and back Turn the user into JSON and then read it back into another variable.

user = { name: "John Smith", age: 35}

```
Ans;   <script>

        let user = { name: "John Smith", age: 35 };

        // Convert the object to JSON string
```

```
        let userJSON = JSON.stringify(user);
        console.log(userJSON); // Output:
{"name":"John Smith","age":35}

        // Parse the JSON string back into an object
        let newUser = JSON.parse(userJSON);
        console.log(newUser); // Output: { name: "John
Smith", age: 35 }


    </script>
```

Document, Event and Controls

Q.1; Create a program to hide/show the password

Ans.
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Show Alert</title>
    <style>
        #passwordInput {
            margin-bottom: 10px;
        }
    </style>
</head>
```

```
<body>
    <label for="password">Password:</label>
    <input type="password" id="passwordInput">
    <input type="checkbox" id="showPasswordCheckbox">
    <label for="showPasswordCheckbox">Show
Password</label>




    <script>

        // Get references to the input field and
checkbox
        const passwordInput =
document.getElementById('passwordInput');
        const showPasswordCheckbox =
document.getElementById('showPasswordCheckbox');

        // Function to toggle password visibility
based on checkbox state
        showPasswordCheckbox.addEventListener('change'
, function () {
            if (showPasswordCheckbox.checked) {
                passwordInput.type = 'text'; // Show
password
            } else {
                passwordInput.type = 'password'; //
Hide password
            }
        });
```

```
    </script>
</body>

</html>
```

Q.2; Create a program that will select all the classes and loop over and whenever i click the button the alert should show

Ans;
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Show Alert</title>
</head>
<body>
    <button id="alertButton">Click Me!</button>
    <div class="example-class">Element 1</div>
    <div class="example-class">Element 2</div>
    <div class="example-class">Element 3</div>


    <script>

        // Get all elements with the specified class
        const elements =
document.querySelectorAll('.example-class');

        // Attach a click event listener to the button
        const alertButton =
document.getElementById('alertButton');
```

```
        alertButton.addEventListener('click', function
() {
            // Loop through each element and show an
alert
            elements.forEach(function (element) {
                alert(element.textContent); // Show
the element's text content in an alert
            });
        });

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

Q.3; Create a responsive header using proper JavaScript

```
Ans; <!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Show Alert</title>
    <style>
        /* CSS for larger screens */
        .menu {
            display: flex;
            list-style: none;
        }

        .menu li {
            margin-right: 20px;
        }
```

```
        /* CSS for smaller screens */
        @media (max-width: 768px) {
            .menu {
                display: none;
            }
        }
    </style>
</head>

<body>
    <header>
        <nav>
            <ul class="menu">
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Services</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
            <button id="toggleMenuButton">Toggle
Menu</button>
        </nav>
    </header>


    <script>

        const toggleMenuButton =
document.getElementById('toggleMenuButton');
        const menu = document.querySelector('.menu');

        toggleMenuButton.addEventListener('click',
function () {
```

```
            // Toggle the menu's display on button
click
            if (menu.style.display === 'none' ||
menu.style.display === '') {
                menu.style.display = 'flex';
            } else {
                menu.style.display = 'none';
            }
        });


    </script>
</body>

</html>
```

Q.4; Create a form and validate using JavaScript

```
Ans; <!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Show Alert</title>

</head>

<body>
    <form id="myForm">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name"
required>
```

```
        <br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email"
required>
        <br>
        <label for="password">Password:</label>
        <input type="password" id="password"
name="password" required>
        <br>
        <button type="submit">Submit</button>
    </form>
    <div id="validationMessage"></div>



    <script>
        const form =
document.getElementById('myForm');
        const validationMessage =
document.getElementById('validationMessage');

        form.addEventListener('submit', function
(event) {
            event.preventDefault(); // Prevent default
form submission

            // Get form values
            const name =
document.getElementById('name').value.trim();
            const email =
document.getElementById('email').value.trim();
            const password =
document.getElementById('password').value.trim();
```

```
            // Validate inputs
            if (name === '' || email === '' ||
password === '') {
                validationMessage.textContent =
'Please fill in all fields';
                return;
            }

            if (!isValidEmail(email)) {
                validationMessage.textContent =
'Please enter a valid email address';
                return;
            }

            // If all validation passes
            validationMessage.textContent = 'Form
submitted successfully!';
            // You can proceed with other actions like
sending the form data to a server

            // Reset form fields
            form.reset();
        });

        // Function to validate email format
        function isValidEmail(email) {
            // Simple email validation using a regular
expression
            const emailRegex =
/^[^\s@]+@[^\s@]+\.[^\s@]+$/;
            return emailRegex.test(email);
        }
```

```
    </script>
</body>

</html>
```

Q.5; Create a modal box using css and Js with three buttons

Ans;
```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document </title>
    <style>
        /* Modal styles */
.modal {
  display: none; /* Hidden by default */
  position: fixed;
  z-index: 1;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5); /* Semi-
transparent background */
}

.modal-content {
  background-color: #fefefe;
  margin: 15% auto;
  padding: 20px;
  border: 1px solid #888;
```

```
  width: 80%;
  max-width: 600px;
}

.close {
  color: #aaa;
  float: right;
  font-size: 28px;
  font-weight: bold;
  cursor: pointer;
}

.close:hover,
.close:focus {
  color: black;
  text-decoration: none;
  cursor: pointer;
}

    </style>

</head>

<body>
    <button id="openModalBtn">Open Modal</button>

    <div id="myModal" class="modal">
      <div class="modal-content">
        <span class="close">&times;</span>
        <p>Modal content goes here.</p>
        <button id="button1">Button 1</button>
        <button id="button2">Button 2</button>
        <button id="button3">Button 3</button>
      </div>
```

```
    </div>



    <script>
      // Get the modal element and buttons
const modal = document.getElementById('myModal');
const openModalBtn =
document.getElementById('openModalBtn');
const closeButton =
document.getElementsByClassName('close')[0];
const buttons =
document.getElementsByClassName('modal-
content')[0].getElementsByTagName('button');

// Function to open the modal
openModalBtn.onclick = function() {
  modal.style.display = 'block';
};

// Function to close the modal
closeButton.onclick = function() {
  modal.style.display = 'none';
};

// Close the modal when clicking outside of it
window.onclick = function(event) {
  if (event.target === modal) {
    modal.style.display = 'none';
  }
};
```

```
// Add click event listeners to the buttons inside the
modal
for (let i = 0; i < buttons.length; i++) {
  buttons[i].addEventListener('click', function() {
    alert(`Button ${i + 1} clicked`);
  });
}


    </script>
</body>

</html>
```

Q.6; Prevent the browser when i click the form submit button

```
Ans; <!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>

</head>

<body>
    <form id="myForm">
        <!-- Your form fields -->
        <input type="text" name="inputField">
        <button type="submit"
id="submitButton">Submit</button>
    </form>
```

```
    <script>

        document.addEventListener('DOMContentLoaded',
function () {
            // Get the form and submit button by their
IDs
            const form =
document.getElementById('myForm');
            const submitButton =
document.getElementById('submitButton');

            // Add an event listener to the form
submission
            form.addEventListener('submit', function
(event) {
                event.preventDefault(); // Prevent
default form submission behavior
                // Your additional logic here, e.g.,
form validation or form data handling
            });

            // Or prevent form submission when the
button is clicked
            submitButton.addEventListener('click',
function (event) {
                event.preventDefault(); // Prevent
default button click behavior
                // Your additional logic here, e.g.,
form validation or form data handling
            });
        });
```

```
    </script>
</body>

</html>
```

New Request

Q.1; What is JSON

Ans; JSON stands for JavaScript Object Notation. It's a lightweight data interchange format that's easy for humans to read and write, and easy for machines to parse and generate. JSON is often used to transmit data between a server and a web application as a text format, and it's based on JavaScript object syntax but is widely used in various programming languages.

JSON is built on two structures:

Key-Value Pairs: JSON data is represented as key-value pairs. The keys are strings, and the values can be strings, numbers, objects, arrays, booleans, or null.

Ordered List of Values: JSON supports arrays, which are ordered lists of values.

Expl.. {

"name": "John Doe",

"age": 30,

"isStudent": false,

"address": {

"street": "123 Main St",

```
    "city": "New York"
  },
  "hobbies": ["reading", "coding", "sports"]
}
```

"name", "age", "isStudent" are keys with string, number, and boolean values.

"address" is a key with an object value containing "street" and "city" keys.

"hobbies" is a key with an array value containing strings.

JSON provides a standard way to represent and exchange data between different systems, making it widely used in web development, especially for APIs (Application Programming Interfaces) where data needs to be transferred between servers and clients. JavaScript provides built-in methods (JSON.parse() and JSON.stringify()) to work with JSON data, allowing conversion between JSON strings and JavaScript objects.

Q,2; What is promises

Ans; Pending: Initial state, neither fulfilled nor rejected.

Fulfilled: The operation completed successfully, and the promise has a value.

Rejected: The operation failed, and the promise has a reason (an error).

Promises are often used for tasks like fetching data from a server, reading files, or handling timeouts. They provide a cleaner alternative to callbacks for handling asynchronous code.

Here's an example of a basic promise:

```
const myPromise = new Promise((resolve, reject) => {
  // Asynchronous operation (e.g., fetching data)
  setTimeout(() => {
    const success = true; // Simulating a successful operation
    if (success) {
```

```
    resolve('Operation completed successfully');
  } else {
    reject('Operation failed');
  }
 }, 2000); // Simulating a delay of 2 seconds
});

// Using the promise
myPromise.then((result) => {
 console.log(result); // Output: "Operation completed successfully"
}).catch((error) => {
 console.error(error); // Output: "Operation failed" (if unsuccessful)
});
```

The Promise constructor takes a callback function with resolve and reject parameters.

Inside the callback, there's an asynchronous operation (simulated with setTimeout).

If the operation succeeds, resolve is called with the successful result. If it fails, reject is called with an error.

then() is used to handle the successful outcome (resolve), and catch() is used to handle errors (reject).

Promises provide a clean way to structure asynchronous code, making it easier to read and maintain compared to deeply nested callbacks (also known as "callback hell"). They also allow chaining multiple asynchronous operations together using .then().

Q.3; Write a program of promises and handle that promises also

Ans; `<script>`

```
        // Simulated function to fetch data from an
API
        function fetchData() {
            return new Promise((resolve, reject) => {
                // Simulating an asynchronous API
request (e.g., fetching data)
                setTimeout(() => {
                    const success = true; //
Simulating a successful operation
                    if (success) {
                        const data = { id: 1, name:
'Example Data' }; // Simulated fetched data
                        resolve(data);
                    } else {
                        reject('Failed to fetch
data');
                    }
                }, 2000); // Simulating a delay of 2
seconds
            });
        }

        // Using the promise
        fetchData()
            .then((result) => {
                console.log('Data fetched:', result);
                // Additional handling of the fetched
data
                return result.name.toUpperCase(); //
Simulating further data processing
            })
            .then((processedData) => {
                console.log('Processed data:',
processedData);
```

```
            })
            .catch((error) => {
                console.error('Error:', error);
            });


    </script>
```

Q.4; Use fetch method for calling an api https://fakestoreapi.com/products

```
 <script>

        fetch('https://fakestoreapi.com/products')
            .then(response => {
                if (!response.ok) {
                    throw new Error('Network response
was not ok');
                }
                return response.json();
            })
            .then(data => {
                console.log('Data from API:', data);
                // Process the fetched data here
            })
            .catch(error => {
                console.error('Error fetching data:',
error);
            });



    </script>
```

Q.5; Display all the product from the api in your HTML page

```
Ans; <!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>product list</title>

</head>

<body>
  <div id="productList"></div>

  <script>
    // Fetch products from the API
    fetch('https://fakestoreapi.com/products')
      .then(response => {
        if (!response.ok) {
          throw new Error('Network response was not
ok');
        }
        return response.json();
      })
      .then(data => {
        // Get the HTML element where you want to
display products
        const productList =
document.getElementById('productList');

        // Create HTML content to display products
        const productsHTML = data.map(product => `
          <div>
            <h2>${product.title}</h2>
```

```
            <p>${product.price}</p>
            <p>${product.description}</p>
            <img src="${product.image}"
alt="${product.title}">
          </div>
        `).join('');

        // Set the HTML content in the productList
element
        productList.innerHTML = productsHTML;
      })
      .catch(error => {
        console.error('Error fetching data:', error);
      });
  </script>
</body>
</html>




</body>

</html>
```

JavaScript Essentials

Q.1; Calculate subtotal price of quantity in JavaScript?

```
Ans; <!DOCTYPE html>

<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Calculate</title>

</head>

<body>
    <form id="orderForm">
        <label for="quantity">Quantity:</label>
        <input type="number" id="quantity" min="1"
value="1"><br>
        <label for="unitPrice">Unit Price:</label>
        <input type="number" id="unitPrice" min="0.01"
step="0.01" value="10.00"><br>
        <button type="button"
onclick="calculateSubtotal()">Calculate
Subtotal</button><br>
        <p id="subtotal">Subtotal: </p>
    </form>


    <script>
        function calculateSubtotal() {
            const quantity =
parseFloat(document.getElementById('quantity').value);
            const unitPrice =
parseFloat(document.getElementById('unitPrice').value)
;

            const subtotal = quantity * unitPrice;
            document.getElementById('subtotal').textCo
ntent = `Subtotal: $${subtotal.toFixed(2)}`;
        }
```

```
    </script>
</body>

</html>




</body>

</html>
```

Q.2; What is JavaScript Output method?

Ans; innerHTML: This property sets or returns the HTML content (inner HTML) of an element. It can be used to dynamically change or update the content of HTML elements.

textContent: This property sets or returns the text content of an element and its descendants. It's often used to update or modify plain text within an HTML element.

console.log(): Although it's primarily used for debugging purposes in development environments, console.log() outputs information to the browser's console. It's not a direct method to display content on a web page but is commonly used to output messages, objects, or variables to the browser's console for debugging purposes.

Alerts and Prompts: JavaScript provides alert() and prompt() functions to display alert boxes and prompt dialogs, respectively, to interact with users.

Writing to the Document: You can directly write content to the document using document.write() method, but it's not commonly used due to potential issues

with overwriting existing content and its behavior when called after the initial document has been loaded.

Q.3; How to used JavaScript Output method?

Ans; innerHTML:

To set the HTML content of an element:

javascript

```
document.getElementById('elementId').innerHTML = '<p>New content</p>';
```

textContent:

To set the text content of an element:

javascript

```
document.getElementById('elementId').textContent = 'Text content';
```

console.log():

To output messages, objects, or variables to the console:

javascript

```
console.log('Message');
console.log(variable);
```

Alerts and Prompts:

alert() to display an alert box with a message:

javascript

```
alert('Alert message');
```

prompt() to display a dialog box prompting the user for input:

javascript

```
let userInput = prompt('Enter something:');
```

Writing to the Document:

document.write() to write directly to the document (use with caution):

javascript

```
document.write('<p>New content</p>');
```

To use these methods effectively, you'll generally select HTML elements using methods like getElementById, getElementsByClassName, or querySelector, and then apply the desired output method to modify or display content within those elements.

For example, you can select an element with an ID and set its text content using textContent:

HTML:

html

```
<p id="output">Initial content</p>
```

JavaScript:

javascript

```
document.getElementById('output').textContent = 'New content';
```

Q.4; How to used JavaScript Events to do all examples?

Ans; Event Handling:

Example: Handling a Button Click Event

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>

</head>

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



    <script>
        document.getElementById('myButton').addEventLis
tener('click', function() {
  // Your action or function to perform on button
click
  alert('Button clicked!');
});
    </script>
</body>

</html>
```

2. Form Submission:

Example: Handling Form Submission Event

```
 <form id="myForm">
        <input type="text" id="inputField">
```

```
        <button type="submit">Submit</button>
      </form>
```

Javascript

```
 <script>
      document.getElementById('myForm').addEventListen
er('submit', function(event) {
  event.preventDefault(); // Prevents default form
submission
  let userInput =
document.getElementById('inputField').value;
  alert(`Submitted: ${userInput}`);
  // Further processing or sending data to a server
});

    </script>
```

3. Input Events:

Example: Handling Input Changes

```
 <input type="text" id="textInput">
```

Javascript

```
 <script>
        document.getElementById('textInput').addEventL
istener('input', function () {
            let textValue = this.value; // Get the
input field value
            console.log(`Input changed:
${textValue}`);
            // Further processing with the changed
input value
```

```
        });


    </script>
```