# **JavaScript Essentials And Advanced**

**(Q.1)** **Write a program to Show an alert**

**(ANS):**

<!DOCTYPE html>

<html>

<head>

<title>Show Alert</title>

</head>

<body>

<button onclick="showAlert ()">Show Alert</button>

<script>

function showAlert () {

alert ("This is an alert message!");

}

</script>

</body>

</html>

**(Q.2) What will be the result for these expressions?**

**1. 5 > 4**

**2. "apple" > "pineapple"**

**3. "2" > "12"**

**4. undefined == null**

**5. undefined === null**

**6. null == "\n0\n"**

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

**(ANS):**

1. 5 > 4: This expression evaluates to true because 5 is greater than 4.

2. "apple" > "pineapple": This expression evaluates to false because in lexicographical order, "apple" comes before "pineapple".

3. "2" > "12": This expression evaluates to true because when comparing strings, JavaScript compares character by character from left to right, and "2" is greater than "1".

4. undefined == null: This expression evaluates to true because in JavaScript, undefined is considered equal to null in loose equality comparison.

5. undefined === null: This expression evaluates to false because strict equality comparison (===) checks both value and type, and undefined and null have different types.

6. null == "\n0\n": This expression evaluates to false because in loose equality comparison, null is only equal to undefined or itself. Here, null is not equal to the string "\n0\n".

7. null === +"\n0\n": This expression evaluates to false. The expression +"\n0\n" coerces the string "\n0\n" to a number, which is 0. However, null is not strictly equal to 0 as they have different types.

**(Q.3) Will alert be shown?**

**if ("0") { alert( 'Hello'); }**

**(ANS):**

Yes, the alert will be shown in JavaScript.

When you have the expression if ("0"), JavaScript evaluates the condition inside the if statement. In this case, "0" is a non-empty string, which is a truthy value in JavaScript. Therefore, the condition evaluates to true, and the code inside the if block, alert('Hello'), will be executed.

So, the alert with the message "Hello" will be shown.

**(Q.4) What is the code below going to output? alert ( null || 2 || undefined );**

**(ANS):**

The code alert (null || 2 || undefined); in JavaScript will output 2.

This is because the || operator in JavaScript returns the first truthy value it encounters when evaluating expressions from left to right.

In this case:

null is false, so it evaluates to false.

2 is truthy, so it evaluates to true, and it stops evaluating further because it has already found a truthy value.

undefined is not evaluated because it comes after a truthy value.

Therefore, the expression results in 2, which will be outputted by the alert () function.

**(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) {**

**if (age> 18) {**

**return true;**

**}**

**else {**

**// ...return confirm (‘did parents allow you?');**

**}**

**}**

**(ANS):**

here is confirmation: -

function checkAge(age) {

if (age > 18) {

return true;

} else {

return confirm ('Did parents allow you?');

}

}

**(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):**

here replace function: -

const ask = (question, yes, no) => {

if (confirm(question)) {

yes ();

} else {

no ();

}

};

ask (

"Do you agree?",

() => {alert ("You agreed.");},

() => {alert ("You canceled the execution.");}

);

**Data Types and Objects**

**(Q.1) Write the code, one line for each action:**

**a) Create an empty object user.**

**b) Add the property name with the value John.**

**c) Add the property surname with the value Smith.**

**d) Change the value of the name to Pete.**

**e) Remove the property name from the object.**

**(ANS):**

A) Create an empty object user.

Ans: - const user = {};

This code declares a variable user and assigns it an empty object literal {}. Now, user is an empty object that you can use to store properties and values.

b) Add the property name with the value John.

Ans: - 1. Using dot notation:

user.name = 'John';

2. Using square bracket notation:

user['name'] = 'John';

Both of these methods will add the name property with the value 'John' to the user object.

c) Add the property surname with the value Smith.

Ans: - user['surname'] = 'Smith';

d) Change the value of the name to Pete.

Ans: - user.name = 'Pete';

e) Remove the property name from the object.

Ans: - 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):**

In JavaScript, arrays are reference types. When you assign an array to another variable, you're not creating a new copy of the array; rather, you're creating a reference to the same array in memory. Therefore, modifications made to the array through one variable will affect the other variable pointing to the same array.

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

let shoppingCart = fruits;

shoppingCart.push("Banana");

alert (fruits. length);

**(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):**

To map the array users to an array containing only the names of the users, you can use the map method along with an arrow function to extract the name property from each object.

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 (', '));

// Output: John, Pete, Mary

**(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):**

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

**(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):**

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

**(Q.6) Destructuring assignment We have an object: Write the Destructuring assignment that**

**reads:**

**a) Name property into the variable name.**

**b) Year’s property into the variable age.**

**c) isAdmin property into the variable isAdmin (false, if no such property)**

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

**(ANS):**

To perform destructuring assignment in JavaScript according to the given instructions:

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

a) Name property into the variable name

let { name } = user;

b) Year’s property into the variable age

let { years: age } = user;

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

let { isAdmin = false } = user;

**(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):**

You can use JSON.stringify() to convert the user object into a JSON string and JSON.parse() to convert the JSON string back into an object.

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

// Convert object to JSON string

let userJSON = JSON.stringify(user);

// Convert JSON string back to object

let userCopy = JSON.parse(userJSON);

console.log(userCopy);

// Output: { name: "John Smith", age: 35 }

**Document, Event and Controls**

**(Q.1) Create a program to hide/show the password**

**(ANS):**

***HTML :-***

<label for="password">Password:</label>

<input type="password" id="password">

<input type="checkbox" id="showPassword">

<label for="showPassword">Show Password</label>

***JAVASCRIPT: -***

const passwordInput = document.getElementById('password');

const showPasswordCheckbox = document.getElementById('showPassword');

showPasswordCheckbox.addEventListener('change', function() {

if (showPasswordCheckbox.checked) {

passwordInput.type = 'text';

} else {

passwordInput.type = 'password';

}

});

**(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):**

***HTML: -***

<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>

***JAVASCRIPT: -***

document.getElementById('alertButton').addEventListener('click', function() {

const elements = document.querySelectorAll('.example-class');

elements.forEach(element => {

alert(element.textContent);

});

});

**(Q.3) Create a responsive header using proper JavaScript**

**(ANS):**

***HTML: -***

<header id="header">

<nav>

<ul>

<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>

</nav>

</header>

***CSS: -***

#header {

background-color: #333;

color: #fff;

}

nav ul {

margin: 0;

padding: 0;

list-style: none;

text-align: center;

}

nav ul li {

display: inline-block;

margin: 0 10px;

}

nav ul li a {

color: #fff;

text-decoration: none;

padding: 10px 20px;

}

***@media screen and (max-width: 600px)*** {

nav ul {

display: none;

}

nav ul.show {

display: block;

}

nav ul.show li {

display: block;

margin: 5px 0;

}

}

***JAVASCRIPT: -***

const header = document.getElementById('header');

const nav = document.querySelector('nav');

const navLinks = document.querySelectorAll('nav ul li');

header.addEventListener('click', function() {

nav.classList.toggle('show');

});

navLinks.forEach(link => {

link.addEventListener('click', function() {

nav.classList.remove('show');

});

});

// Close navigation menu on window resize

window.addEventListener('resize', function() {

if (window.innerWidth > 600) {

nav.classList.remove('show');

}

});

**(Q.4) Create a form and validate using JavaScript**

**(ANS):**

***HTML: -***

<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="error-message" style="color: red;"></div>

***JAVASCRIPT: -***

const form = document.getElementById('myForm');

const errorMessage = document.getElementById('error-message');

form.addEventListener('submit', function(event) {

event.preventDefault(); // Prevent form submission

// Reset error message

errorMessage.textContent = '';

// Validate form fields

const name = form.elements['name'].value.trim();

const email = form.elements['email'].value.trim();

const password = form.elements['password'].value.trim();

if (!name) {

errorMessage.textContent = 'Please enter your name';

return;

}

if (!email) {

errorMessage.textContent = 'Please enter your email';

return;

}

if (!validateEmail(email)) {

errorMessage.textContent = 'Please enter a valid email address';

return;

}

if (!password) {

errorMessage.textContent = 'Please enter your password';

return;

}

// Form is valid, submit it

alert('Form submitted successfully!');

});

// Function to validate email format

function validateEmail(email) {

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

return emailRegex.test(email);

}

**(Q.5) Create a modal box using css and Js with three buttons**

**(ANS):**

***HTML: -***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Modal Box Example</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div id="myModal" class="modal">

<div class="modal-content">

<span class="close">&times;</span>

<p>This is a modal box. Click on a button to close it.</p>

<button id="button1">Button 1</button>

<button id="button2">Button 2</button>

<button id="button3">Button 3</button>

</div>

</div>

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

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

</body>

</html>

***CSS: -***

/\* The Modal (background) \*/

.modal {

display: none; /\* Hidden by default \*/

position: fixed; /\* Stay in place \*/

z-index: 1; /\* Sit on top \*/

left: 0;

top: 0;

width: 100%; /\* Full width \*/

height: 100%; /\* Full height \*/

overflow: auto; /\* Enable scroll if needed \*/

background-color: rgba(0,0,0,0.4); /\* Black w/ opacity \*/

}

/\* Modal Content/Box \*/

.modal-content {

background-color: #fefefe;

margin: 15% auto; /\* 15% from the top and centered \*/

padding: 20px;

border: 1px solid #888;

width: 80%; /\* Could be more or less, depending on screen size \*/

}

/\* The Close Button \*/

.close {

color: #aaa;

float: right;

font-size: 28px;

font-weight: bold;

}

.close:hover,

.close:focus {

color: black;

text-decoration: none;

cursor: pointer;

}

***JAVASCRIPT: -***

// Get the modal

const modal = document.getElementById('myModal');

// Get the button that opens the modal

const btn = document.getElementById('openModalBtn');

// Get the <span> element that closes the modal

const span = document.getElementsByClassName('close')[0];

// When the user clicks the button, open the modal

btn.onclick = function() {

modal.style.display = 'block';

}

// When the user clicks on <span> (x), close the modal

span.onclick = function() {

modal.style.display = 'none';

}

// When the user clicks anywhere outside of the modal, close it

window.onclick = function(event) {

if (event.target == modal) {

modal.style.display = 'none';

}

}

// Button click handlers

document.getElementById('button1').addEventListener('click', function() {

alert('Button 1 clicked');

});

document.getElementById('button2').addEventListener('click', function() {

alert('Button 2 clicked');

});

document.getElementById('button3').addEventListener('click', function() {

alert('Button 3 clicked');

});

**(Q.6) Use external js library to show slider**

**(ANS):**

***HTML: -***

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Slider Example</title>

<!-- Link Slick's CSS -->

<link rel="stylesheet" type="text/css" href="https://cdn.jsdelivr.net/npm/slick-carousel/slick/slick.css"/>

<!-- External CSS for custom styles (if any) -->

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="slider">

<div>Slide 1</div>

<div>Slide 2</div>

<div>Slide 3</div>

<!-- Add more slides as needed -->

</div>

<!-- Link Slick and jQuery -->

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/slick-carousel/slick/slick.min.js"></script>

<!-- External JS for custom scripts (if any) -->

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

</body>

</html>

***JAVASCRIPT: -***

$(document).ready(function(){

$('.slider').slick();

});

**(Q.7) Prevent the browser when i click the form submit button**

**(ANS):**

***HTML: -***

<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" id="submitButton">Submit</button>

</form>

<div id="error-message" style="color: red;"></div>

***JAVASCRIPT: -***

document.getElementById('myForm').addEventListener('submit', function(event) {

event.preventDefault(); // Prevent form submission

// Your form validation code goes here

// Example validation (just for demonstration)

const name = document.getElementById('name').value.trim();

if (!name) {

const errorMessage = document.getElementById('error-message');

errorMessage.textContent = 'Please enter your name';

return;

}

// If form validation passes, you can proceed with other actions

});

**New Request**

**(Q.1) What is JSON**

**(ANS):**

JSON stands for JavaScript Object Notation. It is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON is often used for transmitting data between a server and a web application, as it is language-independent and can be easily parsed by JavaScript, making it ideal for use with AJAX.

JSON data is represented as key-value pairs and structured as a collection of objects (surrounded by curly braces {}) or arrays (surrounded by square brackets []).

**(Q.2) What is promises**

**(ANS):**

Promises are a fundamental concept in JavaScript for handling asynchronous operations. They represent a value that may be available now, in the future, or never. Promises provide a way to deal with asynchronous code in a more synchronous-like manner, making it easier to manage complex asynchronous operations and avoid callback hell.

A Promise can be in one of three states:

Pending: The initial state of a Promise. It represents that the operation represented by the Promise has not yet been fulfilled or rejected.

Fulfilled (Resolved): The state of a Promise when the operation has completed successfully. In this state, the Promise has a resolved value.

Rejected: The state of a Promise when the operation has failed. In this state, the Promise has a reason for rejection, typically an error object.

A Promise is created using the Promise constructor, which takes a function as its argument. This function, known as the executor function, has two parameters: resolve and reject. Inside the executor function, you perform an asynchronous operation, and when it completes, you call resolve if it was successful or reject if it failed.

**(Q.3) Write a program of promises and handle that promises also**

**(ANS):**

// Function to simulate an asynchronous operation

function simulateAsyncOperation() {

return new Promise((resolve, reject) => {

// Simulating an asynchronous operation (e.g., fetching data)

setTimeout(() => {

const randomValue = Math.random();

if (randomValue < 0.5) {

resolve(randomValue); // Operation succeeded

} else {

reject(new Error('Operation failed')); // Operation failed

}

}, 1000); // Simulating a delay of 1 second

});

}

// Consuming the Promise

simulateAsyncOperation()

.then((result) => {

console.log('Success:', result);

})

.catch((error) => {

console.error('Error:', error);

});

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

**(ANS):**

fetch('https://fakestoreapi.com/products')

.then(response => {

// Check if response is successful

if (!response.ok) {

throw new Error('Network response was not ok');

}

// Parse response JSON

return response.json();

})

.then(data => {

// Handle the data received from the API

console.log(data);

})

.catch(error => {

// Handle errors

console.error('There was a problem with the fetch operation:', error);

});

**(Q.5) Display all the product from the api in your HTML page**

**(ANS):**

***HTML: -***

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

</body>

</html>

***CSS: -***

.product-card {

border: 1px solid #ccc;

margin-bottom: 10px;

padding: 10px;

}

***JAVASCRIPT: -***

document.addEventListener('DOMContentLoaded', function() {

const productList = document.getElementById('productList');

// 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(products => {

// Iterate over the products array and create HTML elements for each product

products.forEach(product => {

const productCard = document.createElement('div');

productCard.classList.add('product-card');

const productName = document.createElement('h3');

productName.textContent = product.title;

const productPrice = document.createElement('p');

productPrice.textContent = `$${product.price}`;

const productDescription = document.createElement('p');

productDescription.textContent = product.description;

productCard.appendChild(productName);

productCard.appendChild(productPrice);

productCard.appendChild(productDescription);

productList.appendChild(productCard);

});

})

.catch(error => {

console.error('There was a problem with the fetch operation:', error);

});

});

**JavaScript Essentials**

**(Q.1) What isJavaScript Output method?**

**(ANS):**

JavaScript provides various methods for outputting content to the user, depending on the environment where the JavaScript code is running. Here are some common methods for outputting content in JavaScript:

***1. console.log():*** This method is used to log messages to the console, which is useful for debugging and logging information during development. It's commonly used in web browsers' developer tools and in server-side JavaScript environments like Node.js.

Example:

console.log("Hello, world!");

***2. alert():*** This method displays a dialog box with a specified message and an OK button. It's primarily used for displaying messages to users in web browsers.

Example:

alert("Hello, world!");

***3. document. Write():*** This method writes HTML expressions or JavaScript code to a document. It's commonly used for testing purposes, but it's generally not recommended for regular use due to potential issues with overwriting existing content and performance concerns.

Example:

document. Write("Hello, world!");

***4. innerHTML:*** This property sets or returns the HTML content (inner HTML) of an element. It allows you to dynamically change the content of HTML elements on a webpage.

Example:

document.getElementById("demo").innerHTML = "Hello, world!";

***5. DOM manipulation:*** JavaScript can also manipulate the Document Object Model (DOM) directly to update the content of HTML elements. This involves selecting an HTML element using methods like getElementById(), getElementsByClassName(), or querySelector(), and then modifying its properties or contents.

Example:

<div id="demo"></div>

<script>

document.getElementById("demo").textContent = "Hello, world!";

</script>

**(Q.2) How to used JavaScript Output method?**

**(ANS):**

To use JavaScript output methods, you simply need to call the appropriate method or manipulate the DOM to display content in the desired way. Here's a basic example of how to use each of the mentioned output methods:

***1. console.log():***

console.log("Hello, world!");

This will output "Hello, world!" to the browser's console if you're using a web browser's developer tools or to the terminal if you're using Node.js.

***2. alert() :***

alert("Hello, world!");

This will display an alert dialog with the message "Hello, world!" in the browser.

***3. document.write() ;***

document.write("Hello, world!");

This will write "Hello, world!" directly into the HTML document at the point where the script is executed.

***4. innerHTML:***

document.getElementById("demo").innerHTML = "Hello, world!";

This will set the HTML content of the element with the ID "demo" to "Hello, world!".

***5. DOM manipulation:***

<div id="demo"></div>

<script>

document.getElementById("demo").textContent = "Hello, world!";

</script>

This will set the text content of the div element with the ID "demo" to "Hello, world!".

**(Q.3) How to used JavaScript Events to do all examples?**

**(ANS):**

***1. Using JavaScript events with console.log():***

You can use JavaScript events to trigger a function that logs a message to the console when an event occurs. Here's an example using the click event:

<button onclick="logMessage()">Click me</button>

<script>

function logMessage() {

console.log("Hello, world!");

}

</script>

***2. Using JavaScript events with alert():***

Similarly, you can use JavaScript events to trigger an alert dialog when an event occurs. Here's an example using the click event:

<button onclick="showAlert()">Click me</button>

<script>

function showAlert() {

alert("Hello, world!");

}

</script>

***3. Using JavaScript events with document.write():***

You generally don't use JavaScript events with document.write() as it writes directly to the document and can cause issues with overwriting existing content. However, here's a simple example using the load event:

<script>

window.onload = function() {

document.write("Hello, world!");

};

</script>

***4. Using JavaScript events with innerHTML:***

You can use JavaScript events to dynamically update HTML content using the innerHTML property of an element. Here's an example using the click event:

<button onclick="updateContent()">Click me</button>

<div id="demo"></div>

<script>

function updateContent() {

document.getElementById("demo").innerHTML = "Hello, world!";

}

</script>

***5. Using JavaScript events with DOM manipulation:***

You can use JavaScript events to manipulate the DOM directly. Here's an example using the load event:

<div id="demo"></div>

<script>

window.onload = function() {

document.getElementById("demo").textContent = "Hello, world!";

};

</script>