**ASSIGNMENT JAVASRIPT BASIC & DOM**

**Q.1 What is JavaScript. How to use it?**

**ANS :-**

JavaScript is a versatile programming language primarily used for creating dynamic content on websites. It allows you to manipulate the behavior and appearance of web pages, making them more interactive.

To use JavaScript, you can include it directly in your HTML document using the <script> tag, either within the <head> or <body> section. Here's a basic example:

html

<!DOCTYPE html>

<html>

<head>

<title>My JavaScript Page</title>

<script>

// Your JavaScript code goes here

function greet() {

alert("Hello, world!");

}

</script>

</head>

<body>

<h1>Welcome to my webpage</h1>

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

</body>

</html>

the JavaScript code defines a simple function greet() that displays an alert when the button is clicked.

You can also link to external JavaScript files using the <script> tag with the src attribute:

html

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

**Q.2 How many type of Variable in JavaScript?**

**ANS :-**

In JavaScript, there are three types of variables:

1. var: This is the older way of declaring variables in JavaScript. Variables declared with `var` are function-scoped, and they are hoisted to the top of their function or global scope.

```

javascript

var x = 10;

```

2. let :- Introduced in ECMAScript 6 (ES6), `let` allows you to declare variables with block scope. Variables declared with `let` are not hoisted to the top of the block; they only exist within the block where they are defined.

```

javascript

let y = 20;

```

3. const :- Also introduced in ES6, `const` is used to declare constants. The value of a constant cannot be reassigned once it is initialized. Like `let`, `const` is block-scoped.

```

javascript

const z = 30;

```

These three types of variables provide different levels of scope and flexibility in JavaScript. It's generally recommended to use `const` whenever possible and only use `let` when you need to reassign a variable. The use of “var” is less common in modern JavaScript due to its function-scoping behavior and hoisting quirks.

**Q.3 Define a Data Types in js?**

**ANS :-**

JavaScript has several built-in data types that are used to represent different kinds of values. Here are the main data types in JavaScript:

1. Primitive Data Types :-

- String :- Represents textual data. Example :- `"Hello, World!"`.

- Number :- Represents numeric values. Example :- `42` or `3.14`.

- Boolean :- Represents either `true` or `false`.

- Undefined :- Represents a variable that has been declared but not assigned a value.

- Null :- Represents the intentional absence of any object value.

2. Object Data Type :-

- Object :- Represents a collection of key-value pairs. Objects are used to group related data and functions together.

```

javascript

const person = {

name: "John",

age: 30,

isStudent: false

};

```

3. Special Data Types :-

- Symbol :- Introduced in ECMAScript 6 (ES6), symbols are unique and immutable values that are often used as keys in objects.

```

javascript

const mySymbol = Symbol("uniqueSymbol");

```

4. Composite Data Types :-

- Array :- Represents an ordered collection of values. Arrays can hold values of any data type.

```

javascript

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

```

- Function :- Represents a reusable block of code that can be invoked with a specific set of arguments.

```

javascript

function add(a, b) {

return a + b;

}

```

**Q.4 Write a mul Function Which will Work Properly When invoked With Following Syntax.**

**Ans :-**

Let's create a simple multiplication function called `mul` that can be invoked in various ways:

```

javascript

function mul() {

// Check the number of arguments

const args = Array.from(arguments);

if (args.length === 0) {

// If no arguments are provided, return an error message

return "Error: No arguments provided. Please provide at least two numbers.";

} else if (args.length === 1) {

// If only one argument is provided, return the argument itself

return args[0];

} else {

// If multiple arguments are provided, multiply them together

return args.reduce((product, num) => product \* num, 1);

}

}

// Examples of invoking the mul function

console.log(mul()); // Error: No arguments provided.

console.log(mul(5)); // 5 (returns the single argument)

console.log(mul(2, 3, 4)); // 24 (multiplies all arguments)

console.log(mul(1, 2, 3, 4, 5)); // 120 (multiplies all arguments)

```

This `mul` function checks the number of arguments passed to it and behaves accordingly. It returns an error message if no arguments are provided, returns the single argument if only one is provided, and multiplies all the arguments together if more than one is provided.

**Q.5 What the deference between undefined and undeclare in JavaScript?**

**Ans :-**

In JavaScript, "undefined" and "undeclared" refer to different concepts:

1. Undefined :-

- Defined but not assigned a value :- A variable is "undefined" if it has been declared, but no value has been assigned to it.

```

javascript

let x;

console.log(x); // Outputs: undefined

```

- Property not present in an object :- When you try to access a property that does not exist in an object, the result is "undefined."

```

javascript

const myObj = { name: "John" };

console.log(myObj.age); // Outputs: undefined

```

- Function with no return statement :- If a function does not have a return statement, it implicitly returns "undefined."

```

javascript

function myFunction() {}

console.log(myFunction()); // Outputs: undefined

```

2. Undeclared :-

- Variable not declared :- If you try to access a variable that has not been declared using `var`, `let`, or `const`, you'll get a ReferenceError, indicating that the variable is "undeclared."

```

javascript

console.log(y); // ReferenceError: y is not defined

```

- Strict mode :- In strict mode, attempting to access an undeclared variable results in a ReferenceError.

```

javascript

"use strict";

console.log(z); // ReferenceError: z is not defined

```

**Q.6 Using console.log() print out the following statement: The quote 'There is no exercise better for the heart than reaching down and lifting people up.' by John Holmes teaches us to help one another. Using console.log() print out the following quote by Mother Teresa:**

**Ans :-**

You can print out the provided quote by Mother Teresa using `console.log()` as follows:

```

javascript

console.log("The quote 'There is no exercise better for the heart than reaching down and lifting people up.' by John Holmes teaches us to help one another. - Mother Teresa");

```

This will output:

```

The quote 'There is no exercise better for the heart than reaching down and lifting people up.' by John Holmes teaches us to help one another. - Mother Teresa

```

**Q.7 Check if typeof '10' is exactly equal to 10. If not make it exactly equal?**

**Ans :-**

The `typeof` operator in JavaScript returns a string indicating the type of a variable or operand. In this case, `typeof '10'` will return the string `'string'`, not the number `10`. To check if it's exactly equal to `10` (as a number), you can use a comparison.

Here's how you can do it:

```

javascript

if (typeof '10' === 'number') {

// Do something if '10' is exactly equal to 10 as a number

console.log("'10' is exactly equal to 10 as a number.");

} else {

// Convert '10' to a number if it's not already

const numericValue = parseInt('10', 10); // Using parseInt to convert to a base-10 number

console.log("'10' is not exactly equal to 10 as a number. Converted to:", numericValue);

}

```

**Q.8 Write a JavaScript Program to find the area of a triangle?**

**Ans :-**

The formula to find the area of a triangle is `(base \* height) / 2`. Here's a simple JavaScript program to calculate the area of a triangle:

```

javascript

// Function to calculate the area of a triangle

function calculateTriangleArea(base, height) {

// Check if both base and height are positive numbers

if (base > 0 && height > 0) {

// Calculate the area using the formula

const area = (base \* height) / 2;

return area;

} else {

// If base or height is not a positive number, return an error message

return "Error: Please provide positive values for both base and height.";

}

}

// Example usage

const baseLength = 5;

const triangleHeight = 8;

const areaOfTriangle = calculateTriangleArea(baseLength, triangleHeight);

// Output the result

console.log("The area of the triangle is:", areaOfTriangle);

```

**Q.9 Write a JavaScript program to calculate days left until next Christmas?**

**Ans :-**

Here's a simple JavaScript program to calculate the number of days left until the next Christmas:

```

javascript

// Function to calculate days left until next Christmas

function daysUntilChristmas() {

const currentDate = new Date(); // Get the current date

const currentYear = currentDate.getFullYear(); // Get the current year

// Christmas is always on December 25th, so set the target date to December 25 of the current year

const christmasDate = new Date(currentYear, 11, 25);

// If Christmas has already passed this year, set the target date to December 25 of the next year

if (currentDate > christmasDate) {

christmasDate.setFullYear(currentYear + 1);

}

// Calculate the difference in milliseconds between the current date and Christmas

const timeDifference = christmasDate - currentDate;

// Convert the time difference to days

const daysLeft = Math.ceil(timeDifference / (1000 \* 60 \* 60 \* 24));

return daysLeft;

}

// Example usage

const daysLeftUntilChristmas = daysUntilChristmas();

// Output the result

console.log(`There are ${daysLeftUntilChristmas} days left until next Christmas.`);

```

**Q.10 What is Condition Statement?**

**Ans :-**

A conditional statement in programming is a construct that allows you to make decisions in your code based on certain conditions. It helps your program to execute different blocks of code depending on whether a specified condition evaluates to true or false.

The most common type of conditional statement is the "if" statement. Here's a basic example in JavaScript:

```

javascript

let age = 18;

if (age >= 18) {

console.log("You are eligible to vote");

} else {

console.log("Sorry, you are not eligible to vote");

}

```

In this example, the condition `age >= 18` is evaluated. If it's true, the code inside the curly braces following the `if` statement will be executed; otherwise, the code inside the curly braces following the `else` statement will be executed.

You can also have more complex conditions using "else if" statements to check multiple conditions:

```javascript

let time = 14;

if (time < 12) {

console.log("Good morning!");

} else if (time < 18) {

console.log("Good afternoon!");

} else {

console.log("Good evening!");

}

```

**Q.11 Find circumference of Rectangle formula : C = 4 \* a ?**

**Ans :-**

The formula for the circumference (perimeter) of a rectangle is given by:

\[ C = 2 \times (length + width) \]

Here, \( C \) is the circumference, \( length \) is the length of the rectangle, and \( width \) is the width of the rectangle.

**Q.12 WAP to convert years into days and days into years?**

**Ans :-**

Here's a JavaScript version of a program that converts years into days and days into years:

```

javascript

function yearsToDays(years) {

const days = years \* 365;

return days;

}

function daysToYears(days) {

const years = days / 365;

return years;

}

// Test the functions

const inputYears = 3;

const resultDays = yearsToDays(inputYears);

console.log(`${inputYears} years is equal to ${resultDays} days`);

const inputDays = 1095;

const resultYears = daysToYears(inputDays);

console.log(`${inputDays} days is equal to ${resultYears} years`);

```

**Q.13 Convert temperature Fahrenheit to Celsius? (Conditional logic Question)?**

**Ans :-**

The formula to convert temperature from Fahrenheit to Celsius is:

\[ C = \frac{5}{9} \times (F - 32) \]

Here's a JavaScript program that uses a function to perform this conversion with conditional logic to check if the input is a valid temperature in Fahrenheit:

```javascript

function fahrenheitToCelsius(fahrenheit) {

if (typeof fahrenheit === 'number') {

const celsius = (5 / 9) \* (fahrenheit - 32);

return celsius;

} else {

return "Please enter a valid temperature in Fahrenheit.";

}

}

// Test the function

const inputTemperature = 32;

const resultCelsius = fahrenheitToCelsius(inputTemperature);

console.log(`${inputTemperature}°F is equal to ${resultCelsius.toFixed(2)}°C`);

```

**Q.14 Write a JavaScript exercise to get the extension of a filename.?**

**Ans :-**

Here's a simple JavaScript exercise to get the extension of a filename:

```javascript

function getFileExtension(filename) {

// Check if the filename has a dot (.) indicating an extension

const dotIndex = filename.lastIndexOf('.');

if (dotIndex !== -1) {

// Extract the extension from the filename

const extension = filename.slice(dotIndex + 1);

return extension;

} else {

// If there's no dot, consider it as having no extension

return "No extension found";

}

}

// Test the function

const exampleFilename = "example.txt";

const extensionResult = getFileExtension(exampleFilename);

console.log(`The extension of ${exampleFilename} is: ${extensionResult}`);

```

**Q.15 What is the result of the expression (5 > 3 && 2 < 4)?**

**Ans :-**

The expression `(5 > 3 && 2 < 4)` is a logical AND operation. It checks whether both conditions are true:

- The condition `5 > 3` is true.

- The condition `2 < 4` is also true.

Since both conditions are true, the overall result of the expression is true.

**Q.16 What is the result of the expression (true && 1 && "hello")?**

**Ans:-**

In JavaScript, the logical AND operator (`&&`) evaluates expressions from left to right and stops as soon as it encounters a falsy value. If all values are truthy, it returns the last truthy value.

In the expression `(true && 1 && "hello")`:

- `true` is truthy.

- `1` is truthy.

- `"hello"` is truthy.

Since all values are truthy, the result of the expression is the last truthy value, which is `"hello"`.

**Q.17 What is the result of the expression true && false || false && true?**

**Ans :-**

In JavaScript, the logical AND operator (`&&`) has higher precedence than the logical OR operator (`||`). The expression is evaluated from left to right.

Let's break down the expression `true && false || false && true`:

1. `true && false` evaluates to `false`.

2. `false || false && true` evaluates to `false`.

So, the overall result of the expression is `false`.

**Q.18 What is a Loop and Switch Case in JavaScript define that ?**

**Ans :-**

In JavaScript, a loop is a programming construct that allows you to repeatedly execute a block of code as long as a specified condition is true. There are several types of loops in JavaScript, with the most common ones being the `for` loop, `while` loop, and `do-while` loop.

Here's a basic example of a `for` loop in JavaScript:

```javascript

for (let i = 0; i < 5; i++) {

console.log("Iteration: " + i);

}

```

This loop will execute the code inside the curly braces five times, incrementing the variable `i` from 0 to 4.

Switch case, on the other hand, is a way to handle multiple conditions in a concise and readable manner. It's particularly useful when you have a variable with multiple possible values and you want to execute different code blocks based on its value.

Here's an example of a `switch` statement in JavaScript:

```javascript

let day = "Monday";

switch (day) {

case "Monday":

console.log("It's the start of the week");

break;

case "Wednesday":

console.log("Halfway through the week");

break;

case "Friday":

console.log("Weekend is almost here");

break;

default:

console.log("It's just another day");

}

```

**Q.19 What is the use of is Nan function?**

**Ans :-**

The `isNaN()` function in JavaScript is used to check whether a value is NaN (Not a Number) or not. It returns a boolean value, `true` if the value is NaN, and `false` if the value is a valid number or can be converted to one.

Here's an example:

```javascript

let myValue = "Hello";

if (isNaN(myValue)) {

console.log("The value is not a number or cannot be converted to one.");

} else {

console.log("The value is a valid number.");

}

```

**Q.20 What is the difference between && and || in JavaScript?**

**Ans:-**

In JavaScript, `&&` and `||` are logical operators used for boolean operations.

- `&&` (Logical AND): It returns `true` if both operands are true, and `false` otherwise.

Example:

```javascript

let a = true;

let b = false;

let result = a && b; // result is false

```

- `||` (Logical OR): It returns `true` if at least one of the operands is true, and `false` if both operands are false.

Example:

```javascript

let x = true;

let y = false;

let outcome = x || y; // outcome is true

```

These operators are commonly used in conditional statements and expressions to control the flow of the program based on boolean conditions. For example, in an `if` statement:

```javascript

let age = 25;

if (age >= 18 && age <= 30) {

console.log("You are in the eligible age range");

} else {

console.log("You are not eligible");

}

```

Here, the `&&` operator ensures that both conditions (`age >= 18` and `age <= 30`) must be true for the code inside the `if` block to be executed.

On the other hand, with the `||` operator, the code inside the `if` block would be executed if at least one of the conditions is true.

**Q.21 What is the use of Void (0)?**

**Ans:-**

`void(0)` is often used as a way to produce an undefined value. The `void` operator takes an expression as an operand and evaluates it, but the result is always `undefined`. It's commonly used in scenarios where you want to execute a function or navigate to a URL without affecting the current page.

For example, you might see it used in the `href` attribute of an anchor (`<a>`) element to create a link that doesn't navigate anywhere:

```html

<a href="javascript:void(0);" onclick="myFunction()">Click me</a>

```

In this example, clicking the link will execute the `myFunction()` JavaScript function, and the `void(0)` in the `href` ensures that the page doesn't navigate to a new location. It's a way to prevent the default behavior of the anchor tag without causing any side effects.

**Q.22 Check Number Is Positive or Negative in JavaScript?**

**Ans:-**

You can use a simple `if` statement to check if a number is positive, negative, or zero in JavaScript. Here's an example:

```javascript

function checkNumber(num) {

if (num > 0) {

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

} else if (num < 0) {

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

} else {

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

}

}

// Test the function

checkNumber(5); // Output: The number is positive

checkNumber(-3); // Output: The number is negative

checkNumber(0); // Output: The number is zero

```

This function, `checkNumber`, takes a number as an argument and uses an `if` statement to check whether the number is positive, negative, or zero. You can replace the test values in the function calls with any numeric value you want to check.

**Q.23 Find the Character Is Vowel or Not ?**

**Ans:-**

You can use a simple JavaScript function to check if a character is a vowel or not. Here's an example:

```javascript

function isVowel(char) {

// Convert the character to lowercase for case-insensitivity

const lowerChar = char.toLowerCase();

// Check if the character is a vowel

if (lowerChar === 'a' || lowerChar === 'e' || lowerChar === 'i' || lowerChar === 'o' || lowerChar === 'u') {

return true;

} else {

return false;

}

}

// Test the function

console.log(isVowel('A')); // Output: true

console.log(isVowel('b')); // Output: false

console.log(isVowel('e')); // Output: true

```

In this example, the `isVowel` function takes a character as an argument, converts it to lowercase (for case-insensitivity), and checks if it matches any of the vowels (a, e, i, o, u). The function returns `true` if the character is a vowel and `false` otherwise.

**Q.24 Write to check whether a number is negative, positive or zero?**

**Ans:-**

Here's a simple JavaScript function that checks whether a number is negative, positive, or zero:

```javascript

function checkNumber(num) {

if (num > 0) {

return "Positive";

} else if (num < 0) {

return "Negative";

} else {

return "Zero";

}

}

// Test the function

console.log(checkNumber(5)); // Output: Positive

console.log(checkNumber(-3)); // Output: Negative

console.log(checkNumber(0)); // Output: Zero

```

This function takes a number as an argument and returns a string indicating whether the number is positive, negative, or zero. You can replace the test values in the function calls with any numeric value you want to check.

**Q.25 Write to find number is even or odd using ternary operator in JS?**

**Ans :-**

You can use the ternary operator to determine whether a number is even or odd in JavaScript. Here's an example:

```javascript

function checkEvenOrOdd(number) {

constresult = (number % 2 === 0) ? "Even" : "Odd";

return result;

}

// Test the function

console.log(checkEvenOrOdd(4)); // Output: Even

console.log(checkEvenOrOdd(7)); // Output: Odd

```

In this example, the ternary operator checks if the remainder of the number divided by 2 is equal to 0. If it is, the result is "Even"; otherwise, it's "Odd".

**Q.26 Write find maximum number among 3 numbers using ternary operator in JS?**

**Ans:-**

You can use the ternary operator to find the maximum among three numbers in JavaScript. Here's an example:

```javascript

function findMaxNumber(num1, num2, num3) {

const maxNumber = (num1 > num2) ? ((num1 > num3) ? num1 : num3) : ((num2 > num3) ? num2 : num3);

return maxNumber;

}

// Test the function

console.log(findMaxNumber(8, 15, 5)); // Output: 15

console.log(findMaxNumber(20, 10, 30)); // Output: 30

```

In this example, the ternary operator is used to compare the three numbers and find the maximum among them. The nested ternary operators help in making the comparison for the maximum value.

**Q.27 Write to find minimum number among 3 numbers using ternary operator in JS?**

**Ans :-**

You can use the ternary operator to find the minimum among three numbers in JavaScript. Here's an example:

```

javascript

function findMinNumber(num1, num2, num3) {

const minNumber = (num1 < num2) ? ((num1 < num3) ? num1 : num3) : ((num2 < num3) ? num2 : num3);

return minNumber;

}

// Test the function

console.log(findMinNumber(8, 15, 5)); // Output: 5

console.log(findMinNumber(20, 10, 30)); // Output: 10

```

In this example, the ternary operator is used to compare the three numbers and find the minimum among them. The nested ternary operators help in making the comparison for the minimum value.

**Q.28 Write to find the largest of three numbers in JS?**

**Ans :-**

You can find the largest of three numbers in JavaScript using conditional statements. Here's an example:

```javascript

function findLargestNumber(num1, num2, num3) {

let largestNumber;

if (num1 >= num2 && num1 >= num3) {

largestNumber = num1;

} else if (num2 >= num1 && num2 >= num3) {

largestNumber = num2;

} else {

largestNumber = num3;

}

return largestNumber;

}

// Test the function

console.log(findLargestNumber(8, 15, 5)); // Output: 15

console.log(findLargestNumber(20, 10, 30)); // Output: 30

```

In this example, the function `findLargestNumber` takes three numbers as parameters and uses conditional statements (if-else) to determine which one is the largest.

**Q.29 Write to show**

**i. Monday to Sunday using switch case in JS?**

**ii. Vowel or Consonant using switch case in JS?**

**Ans :-**

Here are examples for both cases using switch statements in JavaScript:

i. Displaying days of the week:

```javascript

function displayDayOfWeek(dayNumber) {

let day;

switch (dayNumber) {

case 1:

day = "Monday";

break;

case 2:

day = "Tuesday";

break;

case 3:

day = "Wednesday";

break;

case 4:

day = "Thursday";

break;

case 5:

day = "Friday";

break;

case 6:

day = "Saturday";

break;

case 7:

day = "Sunday";

break;

default:

day = "Invalid day";

}

console.log(day);

}

// Test the function

displayDayOfWeek(3); // Output: Wednesday

displayDayOfWeek(6); // Output: Saturday

displayDayOfWeek(8); // Output: Invalid day

```

ii. Checking if a character is a vowel or consonant:

```javascript

function checkVowelOrConsonant(char) {

switch (char.toLowerCase()) {

case 'a':

case 'e':

case 'i':

case 'o':

case 'u':

console.log("Vowel");

break;

default:

console.log("Consonant");

}

}

// Test the function

checkVowelOrConsonant('A'); // Output: Vowel

checkVowelOrConsonant('b'); // Output: Consonant

checkVowelOrConsonant('o'); // Output: Vowel

```

In the second example, the switch statement checks if the given character is a vowel or consonant. The `toLowerCase()` method is used to make the comparison case-insensitive.

**• What is JavaScript?**

**Ans :-**

JavaScript is a versatile programming language primarily used for creating interactive effects within web browsers. It allows developers to add dynamic behavior to web pages, making them more engaging and responsive to user actions. JavaScript is a core technology of the World Wide Web alongside HTML and CSS. It is commonly employed for tasks such as form validation, creating animations, fetching data from servers asynchronously (AJAX), building web applications (including single-page applications), and much more. With the advent of server-side JavaScript platforms like Node.js, JavaScript can also be used for server-side programming, enabling developers to create full-stack applications using a single programming language.

**• What is the use of isNaN function?**

**Ans:-**

The `isNaN()` function in JavaScript is used to determine whether a value is NaN (Not-a-Number) or not. NaN is a special value representing the result of an invalid mathematical operation, such as dividing zero by zero or attempting to convert a non-numeric string into a number.

The `isNaN()` function takes a single argument and returns `true` if the argument is NaN, and `false` otherwise. If the argument passed to `isNaN()` is not already of the `number` type, it is first converted to a number before being evaluated.

Here's an example:

```

javascript

isNaN(NaN); // true

isNaN(123); // false

isNaN('Hello'); // true, because 'Hello' cannot be converted to a number

isNaN('123'); // false, because '123' can be converted to the number 123

```

It's important to note that `isNaN()` may produce unexpected results when dealing with non-numeric strings. In such cases, it's often better to use other methods like `Number.isNaN()` introduced in ES6, which doesn't perform type coercion and only returns `true` if the provided value is exactly NaN.

**• What is negative Infinity?**

**Ans:-**

Negative Infinity, denoted as `-Infinity`, is a special value in JavaScript representing the lowest possible numeric value. It is used to represent a value that is smaller than any other numeric value, including negative numbers.

Negative Infinity typically arises as a result of mathematical operations that lead to a value that is smaller than the minimum representable value in JavaScript, or when dividing a negative number by zero.

For example:

```javascript

console.log(Number.NEGATIVE\_INFINITY); // -Infinity

console.log(-1 / 0); // -Infinity

console.log(Number.MAX\_VALUE \* -2); // -Infinity

```

Negative Infinity is often used to represent scenarios where a value tends toward negative infinity, such as in mathematical limits or when an algorithm reaches an extreme negative value. It's important to handle such cases carefully in code to avoid unexpected behavior or errors.

**• Which company developed JavaScript?**

**Ans:-**

JavaScript was developed by Netscape Communications Corporation, specifically by Brendan Eich. It was initially created in 1995 under the name "Mocha" but was later renamed "LiveScript" before finally being called JavaScript.

**• What are undeclared and undefined variables?**

**Ans:-**

Undeclared and undefined variables are two different concepts in JavaScript:

1. Undeclared Variables :- These are variables that have not been declared using the `var`, `let`, or `const` keywords before being used. If you try to use an undeclared variable, JavaScript will raise a `ReferenceError`. For example:

```javascript

// This will throw a ReferenceError because `x` is undeclared

console.log(x);

```

2. Undefined Variables :- These are variables that have been declared but have not been assigned a value, or variables that have been explicitly assigned the value `undefined`. When you access such variables, they return the value `undefined`. For example:

```javascript

let y;

console.log(y); // Output will be: undefined

let z = undefined;

console.log(z); // Output will also be: undefined

```

**• Write the code for adding new elements dynamically?**

**Ans:-**

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Add Elements Dynamically</title>

</head>

<body>

<h2>Add Elements Dynamically</h2>

<div id="container">

<!-- Existing elements -->

<p>Existing Element 1</p>

<p>Existing Element 2</p>

</div>

<button onclick="addElement()">Add Element</button>

<script>

function addElement() {

// Create the HTML for the new element

var newElementHTML = "<p>New Element</p>";

// Get the container element

var container = document.getElementById("container");

// Concatenate the new HTML with the existing HTML of the container

container.innerHTML += newElementHTML;

}

</script>

</body>

</html>

```

In this example, the `addElement()` function creates the HTML string for the new element, then appends this HTML to the existing HTML content of the container element by directly modifying its `innerHTML` property. This approach achieves the same result as using `appendChild()` but without explicitly calling it.

**• What is the difference between ViewState and SessionState?**

**Ans:-**

ViewState and SessionState are both mechanisms used in ASP.NET for state management, but they serve different purposes and have different scopes:

1. ViewState :-

- ViewState is used to preserve the state of a page and its controls across postbacks. It stores the state of individual controls and is maintained on the client-side as a hidden field within the HTML output of the page.

- ViewState is specific to a single web page and is used to persist data between round-trips to the server and back to the client. It helps in maintaining the state of controls such as textboxes, checkboxes, dropdown lists, etc., so that their values remain consistent across postbacks.

- ViewState is lightweight and suitable for storing small amounts of data related to a single page.

2. SessionState :-

- SessionState, on the other hand, is used to store user-specific data across multiple requests and pages during a user session.

- It is stored on the server-side and can store larger amounts of data compared to ViewState.

- SessionState is accessible across multiple pages within the same web application and is often used to maintain user authentication, user preferences, shopping cart contents, etc.

- SessionState data is scoped to a user's session and can be accessed by any page within that session.

**• What is === operator?**

**Ans :-**

The `===` operator in JavaScript is known as the "strict equality operator." It is used to compare two values for equality without performing type coercion. This means that both the value and the type of the operands must be the same for the `===` operator to return `true`.

Here's how it works:

- If the operands are of the same type and have the same value, `===` returns `true`.

- If the operands are of different types or have different values, `===` returns `false`.

For example:

```javascript

5 === 5; // true, both the value and type are the same

5 === '5'; // false, different types (number vs. string)

0 === false; // false, different types and values

```

In contrast to the `===` operator, the `==` operator performs type coercion before comparing the values. This means that if the operands are of different types, JavaScript will attempt to convert them to a common type before making the comparison. This can sometimes lead to unexpected results.

For example, `0 == false` evaluates to `true` because JavaScript converts `false` to `0` before performing the comparison. However, `0 === false` evaluates to `false` because they are of different types.

Using `===` is generally recommended for comparisons in JavaScript because it avoids the potential pitfalls of type coercion and leads to more predictable behavior.

**• How can the style/class of an element be changed?**

**Ans:-**

You can change the style or class of an HTML element using JavaScript. Here's how you can do it:

1. Changing Style :-

You can change the CSS style of an element directly by modifying its `style` property. This property allows you to manipulate individual CSS properties of an element. For example:

```javascript

// Get the element by its ID

var element = document.getElementById("myElement");

// Change its background color

element.style.backgroundColor = "blue";

// Change its font size

element.style.fontSize = "20px";

```

2. Changing Class :-

You can also change the class of an element by modifying its `className` or `classList` property. The `className` property replaces the entire class attribute of the element, while the `classList` property provides more flexibility for adding, removing, and toggling individual classes. For example:

```javascript

// Get the element by its ID

var element = document.getElementById("myElement");

// Change its class using className

element.className = "newClass";

// Change its class using classList (add, remove, toggle)

element.classList.add("newClass");

element.classList.remove("oldClass");

element.classList.toggle("active");

```

Using `classList` is generally preferred because it provides more convenient methods for manipulating classes and avoids common issues with directly manipulating the `className` property, such as accidentally overriding existing classes.

By using JavaScript to manipulate styles and classes dynamically, you can create interactive and responsive web pages that adapt to user actions and events.

**• How to read and write a file using JavaScript?**

**Ans :-**

In a web browser environment, JavaScript alone cannot directly read or write files on a user's system for security reasons. However, it can interact with files indirectly through user actions such as file input in HTML forms or by utilizing Web APIs like the File API or the FileReader API.

Here's a basic example of reading a file using JavaScript with the FileReader API:

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>File Reader Example</title>

</head>

<body>

<input type="file" id="fileInput">

<button onclick="readFile()">Read File</button>

<script>

function readFile() {

var fileInput = document.getElementById('fileInput');

var file = fileInput.files[0];

var reader = new FileReader();

reader.onload = function(event) {

var contents = event.target.result;

console.log(contents); // Display the contents of the file

};

reader.onerror = function(event) {

console.error("File could not be read! Code " + event.target.error.code);

};

reader.readAsText(file);

}

</script>

</body>

</html>

```

This code creates an input element of type "file" and a button. When the button is clicked, it triggers the `readFile()` function. Inside this function, the FileReader API is used to read the contents of the selected file. The `readAsText()` method is used to read the file as a text file. Once the file is successfully read, the `onload` event is triggered, and the contents of the file are logged to the console.

For writing files, JavaScript in a web browser environment does not have direct access to write files to the user's system due to security restrictions. However, you can generate downloadable files by creating data URLs or using Blob objects. Alternatively, in server-side environments like Node.js, you can use the built-in File System module to read and write files.

**• What are all the looping structures in JavaScript?**

**Ans :-**

JavaScript offers several looping structures to execute a block of code repeatedly. The main looping structures in JavaScript are:

1. \*\*for loop\*\*: The `for` loop is used to execute a block of code a specified number of times. It consists of three parts: initialization, condition, and iteration. Syntax:

```javascript

for (initialization; condition; iteration) {

// code block to be executed

}

```

2. \*\*while loop\*\*: The `while` loop executes a block of code as long as a specified condition is true. Syntax:

```javascript

while (condition) {

// code block to be executed

}

```

3. \*\*do...while loop\*\*: Similar to the `while` loop, the `do...while` loop executes a block of code once, and then repeats the loop as long as a specified condition is true. Syntax:

```javascript

do {

// code block to be executed

} while (condition);

```

4. \*\*for...in loop\*\*: The `for...in` loop iterates over the properties of an object. Syntax:

```javascript

for (variable in object) {

// code block to be executed

}

```

5. \*\*for...of loop\*\*: The `for...of` loop is used to iterate over iterable objects such as arrays, strings, maps, sets, etc. Syntax:

```javascript

for (variable of iterable) {

// code block to be executed

}

```

These looping structures provide different ways to iterate over data or execute a block of code repeatedly, allowing developers to choose the most appropriate one based on the requirements of their code.

**• How can you convert the string of any base to an integer in JavaScript?**

**Ans:-**

To convert a string representation of a number in any base to an integer in JavaScript, you can use the `parseInt()` function along with specifying the radix (base) as the second argument. The radix indicates the base of the numeral system used in the string. Here's how you can do it:

```javascript

// Convert a binary string to an integer

var binaryString = "1010";

var integerValue = parseInt(binaryString, 2);

console.log(integerValue); // Output: 10

// Convert a hexadecimal string to an integer

var hexString = "1A";

var integerValue = parseInt(hexString, 16);

console.log(integerValue); // Output: 26

// Convert an octal string to an integer

var octalString = "23";

var integerValue = parseInt(octalString, 8);

console.log(integerValue); // Output: 19

```

In each case, `parseInt()` takes the string representation of a number (`binaryString`, `hexString`, `octalString`) and converts it to an integer based on the specified radix (`2`, `16`, `8` for binary, hexadecimal, and octal respectively). The function returns the integer value.

**• What is the function of the delete operator?**

**Ans :-**

The `delete` operator in JavaScript is used to remove a property from an object. It allows you to delete both own properties of an object and properties inherited from its prototype chain. Here's how the `delete` operator works:

```javascript

var obj = { a: 1, b: 2, c: 3 };

// Deleting a property

delete obj.b;

console.log(obj); // Output: { a: 1, c: 3 }

```

In this example, the property `b` is removed from the object `obj` using the `delete` operator. After deletion, the object only contains the properties `a` and `c`.

It's important to note that `delete` only affects the properties of an object, not the object itself. It returns `true` if the operation is successful (i.e., if the property is successfully deleted), and `false` otherwise. However, there are some limitations to using `delete`:

- `delete` only works on properties with configurable attributes. Non-configurable properties (those created using `Object.defineProperty()` with `configurable: false`) cannot be deleted.

- `delete` does not remove properties that are declared with `var`, `let`, or `const` keywords or function declarations.

- `delete` does not affect variables or functions declared in the global scope or variables declared with `var` within functions.

The `delete` operator is most commonly used when you need to dynamically remove properties from an object, such as when manipulating complex data structures or managing object properties at runtime.

**• What are all the types of Pop up boxes available in JavaScript?**

**Ans:-**

JavaScript provides several types of popup boxes for displaying messages or prompting user input. The main types of popup boxes are:

1. \*\*Alert Box\*\*: Displays a message to the user with an "OK" button. It's used to provide information or alerts.

```javascript

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

```

2. \*\*Confirm Box\*\*: Displays a message to the user with "OK" and "Cancel" buttons. It's used to confirm or cancel an action.

```javascript

var result = confirm("Are you sure you want to proceed?");

if (result === true) {

// User clicked OK

} else {

// User clicked Cancel

}

```

3. \*\*Prompt Box\*\*: Displays a message to the user with an input field where they can enter data. It's used to prompt the user for input.

```javascript

var userInput = prompt("Please enter your name:", "John Doe");

if (userInput !== null) {

// User entered something

} else {

// User clicked Cancel

}

```

**• What is the use of Void (0)?**

**Ans:-**

In JavaScript, `void(0)` is a special expression that evaluates to `undefined`. It is often used in the context of anchor (`<a>`) elements in HTML to prevent the browser from navigating to a new page when the anchor is clicked without having to specify an actual URL.

Here's how it's typically used:

```html

<a href="javascript:void(0)">Click me</a>

```

When a user clicks on the "Click me" link, the browser will execute the JavaScript expression `void(0)`, which returns `undefined`. Since there is no valid URL associated with the anchor, the browser stays on the same page without navigating anywhere.

The use of `void(0)` is a common practice in situations where you want to attach JavaScript behavior to an anchor element without causing any navigation. However, it's worth noting that modern JavaScript practices tend to favor more semantic and accessible solutions, such as using event listeners or leveraging the `href` attribute with a meaningful URL or `#` for anchor links that trigger JavaScript actions.

**• How can a page be forced to load another page in JavaScript?**

**Ans :-**

In JavaScript, you can force a page to load another page by setting the `window.location` property to the URL of the desired page. This triggers a navigation to the specified URL, effectively loading a new page. Here's how you can do it:

```javascript

window.location = "https://www.example.com/newpage.html";

```

When this line of code is executed, the browser will navigate to the URL specified in the `window.location` assignment, causing the current page to be unloaded and the new page to be loaded.

You can also use other methods of the `window.location` object to perform different types of navigation, such as reloading the current page (`window.location.reload()`), navigating to the previous page in the browser's history (`window.history.back()`), or navigating to the next page in the browser's history (`window.history.forward()`).

It's important to note that forcing a page to load another page abruptly can disrupt the user experience and might be considered bad practice in certain contexts. Ensure that such navigation actions are clear and expected by users.

**• What are the disadvantages of using innerHTML in JavaScript?**

**Ans :-**

While the `innerHTML` property in JavaScript is convenient for manipulating the content of HTML elements, it also has some disadvantages:

1. Security Risks :- Using `innerHTML` can expose your application to cross-site scripting (XSS) attacks if the content being inserted contains untrusted or user-generated data. If this data includes malicious scripts, they can be executed within your page, leading to security vulnerabilities.

2. Performance Overhead :- Manipulating `innerHTML` often involves parsing and re-rendering the entire content of the targeted element, which can be inefficient for large or complex DOM structures. This can result in performance overhead, especially on mobile devices or older browsers.

3. Event Handler Removal :- When you set `innerHTML` to replace the content of an element, any event listeners attached to child elements will be lost. This requires re-attaching event listeners after updating the content, which can be cumbersome and error-prone.

4. Loss of References :- If the content being replaced or inserted via `innerHTML` includes elements with JavaScript event handlers or references, those references may be lost or become invalid after the content is replaced, leading to unexpected behavior or memory leaks.

5. Non-Standard HTML Parsing :- While most modern browsers handle `innerHTML` consistently, there can be differences in how HTML is parsed and rendered across different browsers. This can lead to inconsistencies or unexpected behavior in edge cases.

To mitigate these disadvantages, consider using alternative methods for DOM manipulation, such as `createElement`, `appendChild`, `insertBefore`, or more advanced techniques like using virtual DOM libraries (e.g., React, Vue.js) or server-side templating engines (e.g., Handlebars, EJS) for building dynamic user interfaces. Additionally, always sanitize and validate any user-generated content before inserting it into the DOM to prevent XSS attacks.

**• Create password field with show hide functionalities**

**Ans:-**

You can create a password field with show/hide functionality using HTML, CSS, and JavaScript. Here's a simple example:

HTML:

```html

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

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

<span id="togglePassword" onclick="togglePasswordVisibility()">Show</span>

```

CSS:

```css

#togglePassword {

cursor: pointer;

color: blue;

text-decoration: underline;

}

```

JavaScript:

```javascript

function togglePasswordVisibility() {

var passwordField = document.getElementById("password");

var toggleButton = document.getElementById("togglePassword");

if (passwordField.type === "password") {

passwordField.type = "text";

toggleButton.textContent = "Hide";

} else {

passwordField.type = "password";

toggleButton.textContent = "Show";

}

}

```

In this example, we have an HTML input field of type "password" for entering passwords. Next to it, we have a span element with the id "togglePassword" that acts as a button to toggle the password visibility. When the button is clicked, the `togglePasswordVisibility()` function is called, which toggles the type of the password field between "password" and "text" (to show or hide the password), and updates the text content of the button accordingly.

This provides a simple way for users to toggle the visibility of the password they are entering.