Web Designing Assignment

Module - 4 (JAVASCRIPT BASIC & DOM)

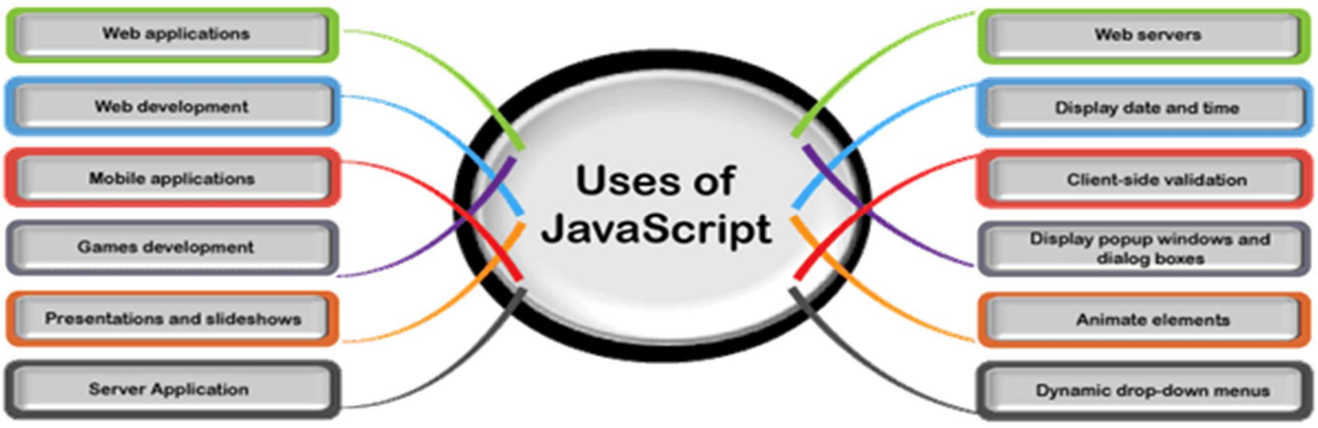
(Basic logic Question)

1. What is JavaScript. How to use it?

JavaScript is a light-weight object-oriented programming language that is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language. JavaScript enables dynamic interactivity on websites when it is applied to an HTML document.

JavaScript helps the users to build modern web applications to interact directly without reloading the page every time. JavaScript is commonly used to dynamically modify HTML and CSS to update a user interface by the DOM API. It is mainly used in web applications.

Let's discuss the uses of JavaScript. Some of the uses of JavaScript are representing in the following image.



1. Web Applications

As day-by-day there is a continuous improvement in the browsers, so JavaScript gained popularity for making robust web applications. We can understand it by taking the example of **Google Maps**. In Maps user just requires to click and drag the mouse; the details are visible just by a click. There is a use of JavaScript behind these concepts.

2. Web Development

JavaScript is commonly used for creating web pages. It allows us to add dynamic behavior to the webpage and add special effects to the webpage. On websites, it is mainly used for validation purposes. JavaScript helps us to execute complex actions and also enables the interaction of websites with visitors. Using JavaScript, it is also possible to load the content in a document without reloading the webpage.

3. Mobile Applications

Now a day's mobile devices are broadly used for accessing the internet. Using JavaScript, we can also build an application for non-web contexts. The features and uses of JavaScript make it a powerful tool for creating mobile applications. The **React Native** is the widely used JavaScript framework for creating mobile applications. Using **React Native,** we can build mobile applications for different operating systems. We do not require writing different codes for the iOS and Android operating systems. We only need to write it once and run it on different platforms.

4. Game

JavaScript is also used for creating games. It has various libraries and frameworks for creating a game. The game can either be a 2D or 3D. Some JavaScript game engines such as **PhysicsJS, Pixi.js** help us to create a web game. We can also use the **WebGL** (web graphics library), which is the JavaScript API to render 2D and 3D images on browsers.

5. Presentations

JavaScript also helps us to create presentations as a website. The libraries, such as **RevealJs,** and **BespokeJs,** can be used to create a web-based slide deck. They are easier to use, so we can easily make something amazing in a short time.

The **Reveal.js** is used to create interactive and beautiful slide decks with the help of HTML. These presentations work great with mobile devices and tablets. It also supports all of the CSS color formats. The **BespokeJS** includes animated bullet lists, responsive scaling, and a wide variety of features.

6. Server Applications

A large number of web applications have a server-side to them. JavaScript is used to generate content and handle HTTP requests. JavaScript can also run on servers through **Node.js**.

The Node.js provides an environment containing the necessary tools required for JavaScript to run on servers.

7. Web Servers

A web server can be created by using **Node.js**. Node.js is event-driven and not waits for the response of the previous call. The servers created using Node.js are fast and don't use buffering and transfer chunks of data. The HTTP module can be used to create the server by using the **createServer()** method. This method executes when someone tries to access the port 8080. As a response, the HTTP server should display HTML and should be included in the HTTP header.

In this article, we discussed various JavaScript applications. JavaScript has various other uses that help us to improve the performance of webpages. The other uses of JavaScript are listed as follows:

- Client-side validation.
- Displaying date and time.
- To validate the user input before submission of the form.
- Open and close new windows.
- To display dialog boxes and pop-up windows.
- To change the appearance of HTML documents.
- To create the forms that respond to user input without accessing the server.

2. How many type of Variable in JavaScript?

In JavaScript, variables are used to store and manage data. They are created using the var, let, or const keyword.

var Keyword

The var keyword is used to declare a variable. It has a function-scoped or globally-scoped behavior.

```
var x = 10;
```

let Keyword

The let keyword is a block-scoped variables. It's commonly used for variables that may change their value.

```
let y = "Hello";
```

const Keyword
The const keyword declares variables that cannot be reassigned. It's block-scoped as well.

```
const PI = 3.1
```

3. Define a Data Types in js?

JavaScript, is a **dynamically typed** (also called loosely typed) scripting language. In JavaScript, variables can receive different data types over time.

The latest ECMAScript standard defines eight data types Out of which seven data types are **Primitive(predefined)** and one **complex or Non-Primitive**.

Primitive Data Types:

The predefined data types provided by JavaScript language are known as primitive data types. Primitive data types are also known as in-built data types.

- **Number:** JavaScript numbers are always stored in double-precision 64-bit binary format IEEE 754. Unlike other programming languages, you don't need int, float, etc to declare different numeric values.
- **String:** JavaScript Strings are similar to sentences. They are made up of a list of characters, which is essentially just an "array of characters, like "Hello GeeksforGeeks" etc.
- **Boolean:** Represent a logical entity and can have two values: true or false.
- **Null:** This type has only one value that is *null.*
- **Undefined:** A variable that has not been assigned a value is *undefined.*
- **Symbol:** Symbols return unique identifiers that can be used to add unique property keys to an object that won't collide with keys of any other code that might add to the object.
- **BigInt:** BigInt is a built-in object in JavaScript that provides a way to represent whole numbers larger than 253-1.

Non-Primitive Data Types:

The data types that are derived from primitive data types of the JavaScript language are known as non-primitive data types. It is also known as derived data types or reference data types.

- **Object:** It is the most important data type and forms the building blocks for modern JavaScript. We will learn about these data types in detail in further articles.

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

The **MUL function** is a miniature of the multiplication function. In this function, we call the function that required an argument as a first number, and that function calls another function that required another argument and this step goes on.

The first function’s argument is x, the second function`s argument is y and the third is z, so the return value will be xyz.

Syntax:

```
function mul(x) {
 return function (y) {
  return function (z) {
   return x * y * z;
  };
 };
}
```

Example: Below example illustrates the MUL() function in JavaScript.

```
<script>
  function mul(x) {
  return function(y) {
    return function(z) {
    return x*y*z;
    };
  }
  }
  console.log(mul(2)(3)(5));
  console.log(mul(2)(3)(4));
</script>
```

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

Undefined: It occurs when a variable has been declared but has not been assigned any value. Undefined is not a keyword.

Undeclared: It occurs when we try to access any variable that is not initialized or declared earlier using the var or const keyword.

S.No.	*Undeclared*	*Undefined*
1.	These are the variables that do not exist in the memory heap.	These variables are the ones that do exist in memory but nothing is being assigned to them explicitly by the programmer.
2.	The variables are considered to be undeclared because of programmer does not write them with var, let, or const.	The variables are considered to be undefined because it is assigned by javascript to them.
3.	If we try to access them in the code execution phase then javascript will throw a Reference error.	If we try to access these variables we'll get the undefined as value.

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:**

   Check program

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

   Check program

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

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

Check program

10. What is Condition Statement?

1. Using if Statement

The if statement is used to evaluate a particular condition. If the condition holds true, the associated code block is executed.

Syntax:

```
if ( condition ) {
   // If the condition is met,
   //code  will get executed.
}
```

Example: In this example, we are using the if statement to find given number is even or odd.

```
let num = 20;

if (num % 2 === 0) {
   console.log("Given number is even number.");
}

if (num % 2 !== 0) {
   console.log("Given number is odd number.");
};
```

Output

```
Given number is even number.
```

Explanation: This JavaScript code determines if the variable `num` is even or odd using the modulo operator `%`. If `num` is divisible by 2 without a remainder, it logs “Given number is even number.” Otherwise, it logs “Given number is odd number.”

2. Using if-else Statement

The if-else statement will perform some action for a specific condition. Here we are using the else statement in which the else statement is written after the if statement and it has no condition in their code block.

Syntax:

```
if (condition1) {
   // Executes when condition1 is true
   if (condition2) {
      // Executes when condition2 is true
   }
}
```

Example: In this example, we are using if-else conditional statement to check the driving licence eligibility date.

```
let age = 25;

if (age >= 18) {
   console.log("You are eligible of driving licence")
} else {
   console.log("You are not eligible for driving licence")
};
```

Output

```
You are eligible of driving licence
```

Explanation: This JavaScript code checks if the variable `age` is greater than or equal to 18. If true, it logs “You are eligible for a driving license.” Otherwise, it logs “You are not eligible for a driving license.” This indicates eligibility for driving based on age.

3. else if Statement

The else if statement in JavaScript allows handling multiple possible conditions and outputs, evaluating more than two options based on whether the conditions are true or false.

Syntax:

```
if (1st condition) {
   // Code for 1st condition
} else if (2nd condition) {
   // ode for 2nd condition
} else if (3rd condition) {
   // Code for 3rd condition
} else {
   //  ode that will execute if all above conditions are false
}
```

Example: In this example, we are using the above-explained approach.

```
const num = 0;

if (num > 0) {
   console.log("Given number is positive.");
} else if (num < 0) {
   console.log("Given number is negative.");
} else {
   console.log("Given number is zero.");
};
```

Output

```
Given number is zero.
```

Explanation: This JavaScript code determines whether the constant `num` is positive, negative, or zero. If `num` is greater than 0, it logs “Given number is positive.” If `num` is less than 0, it logs “Given number is negative.” If neither condition is met (i.e., `num` is zero), it logs “Given number is zero.”

4. Using Switch Statement (JavaScript Switch Case)

As the number of conditions increases, you can use multiple else-if statements in JavaScript. but when we dealing with many conditions, the switch statement may be a more preferred option.

Syntax:

```
switch (expression) {
   case value1:
      statement1;
      break;
   case value2:
      statement2;
      break;
   . . .
   case valueN:
      statementN;
      break;
   default:
      statementDefault;
};
```

Example: In this example, we find a branch name Based on the student's marks, this switch statement assigns a specific engineering branch to the variable Branch. The output displays the student's branch name,

```
const marks = 85;

let Branch;

switch (true) {
    case marks >= 90:
        Branch = "Computer science engineering";
        break;
    case marks >= 80:
        Branch = "Mechanical engineering";
        break;
    case marks >= 70:
        Branch = "Chemical engineering";
        break;
    case marks >= 60:
        Branch = "Electronics and communication";
        break;
    case marks >= 50:
        Branch = "Civil engineering";
        break;
    default:
        Branch = "Bio technology";
        break;
}

console.log(`Student Branch name is : ${Branch}`);
```

Output

```
Student Branch name is : Mechanical engineering
```

Explanation:
This JavaScript code assigns a branch of engineering to a student based on their marks. It uses a switch statement with cases for different mark ranges. The student's branch is determined according to their marks and logged to the console.

5. Using Ternary Operator (?:)

The conditional operator, also referred to as the ternary operator (?:), is a shortcut for expressing conditional statements in JavaScript.

Syntax:

```
condition ? value if true : value if false
```

Example: In this example, we use the ternary operator to check if the user's age is 18 or older. It prints eligibility for voting based on the condition.

```
let age = 21;

const result =
   (age >= 18) ? "You are eligible to vote."
      : "You are not eligible to vote.";

console.log(result);
```

Output

```
You are eligible to vote.
```

Explanation: This JavaScript code checks if the variable `age` is greater than or equal to 18. If true, it assigns the string "You are eligible to vote." to the variable `result`. Otherwise, it assigns "You are not eligible to vote." The value of `result` is then logged to the console.

6. Nested if...else

Nested if…else statements in JavaScript allow us to create complex conditional logic by checking multiple conditions in a hierarchical manner. Each if statement can have an associated else block, and within each if or else block, you can nest another if…else statement. This nesting can continue to multiple levels, but it's important to maintain readability and avoid excessive complexity.

Syntax:

```
if (condition1) {
   // Code block 1
   if (condition2) {
      // Code block 2
   } else {
      // Code block 3
   }
} else {
   // Code block 4
}
```

Example: This example demonstrates how nested if...else statements can be used to handle different scenarios based on multiple conditions.

```
let weather = "sunny";
let temperature = 25;

if (weather === "sunny") {
   if (temperature > 30) {
      console.log("It's a hot day!");
   } else if (temperature > 20) {
      console.log("It's a warm day.");
   } else {
      console.log("It's a bit cool today.");
   }
} else if (weather === "rainy") {
   console.log("Don't forget your umbrella!");
} else {
   console.log("Check the weather forecast!");
};
```

Output

```
It's a warm day.
```

Explanation: In this example, the outer if statement checks the weather variable. If it's "sunny," it further checks the temperature variable to determine the type of day it is (hot, warm, or cool). Depending on the values of weather and temperature, different messages will be logged to the console.

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

Check program

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

Check program

13.Convert temperature Fahrenheit to Celsius?

Check program

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

Check program

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

The result is true, because both conditions are true.

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

The result is "hello", because all the operands are truthy, and the "&&" operator returns the last truthy operand.

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

The result is false, because the "&&" operator has higher precedence than the "||" operator, so the expression is equivalent to ((true && false) || (false && true)), which returns false.

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

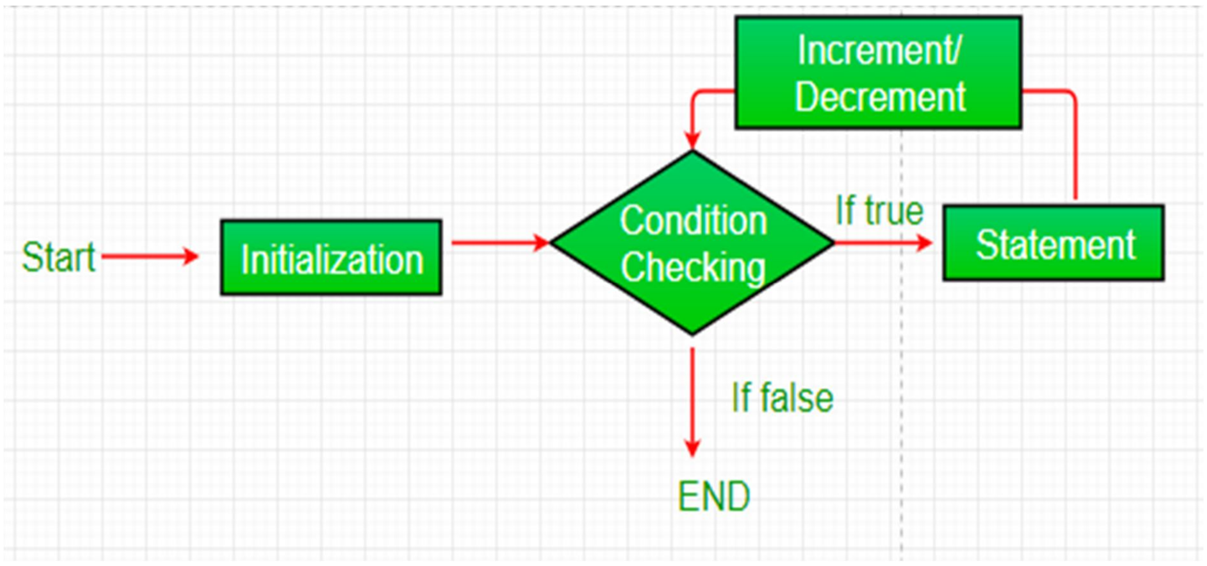
❖ JavaScript for Loop

The JS for loop provides a concise way of writing the loop structure. The for loop contains initialization, condition, and increment/decrement in one line thereby providing a shorter, easy-to-debug structure of looping.

Syntax

```
for (initialization; testing condition; increment/decrement) {
   statement(s)
}
```

Flowchart



- **Initialization condition:** It initializes the variable and mark the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.
- **Test Condition:** It is used for testing the exit condition of a for loop. It must return a boolean value. It is also an **Entry Control Loop** as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to be true, the statements in the loop body are executed.
- **Increment/ Decrement:** It is used for updating the variable for the next iteration.
- **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

Example

```
// JavaScript program to illustrate for loop
let x;
// for loop begins when x = 2
// and runs till x <= 4
for (x = 2; x <= 4; x++) {
    console.log("Value of x: " + x);
}
```

Output

```
Value of x: 2
Value of x: 3
Value of x: 4
```

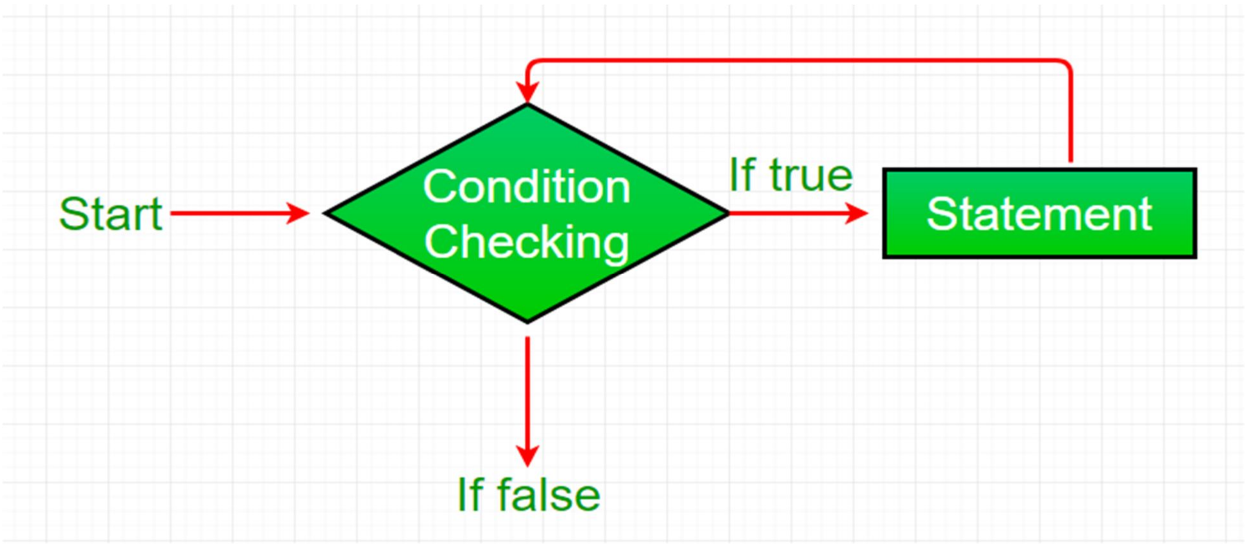
❖ JavaScript while Loop

The JS while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax

```
while (boolean condition) {
   loop statements...
}
```

Flowchart



- While loop starts with checking the condition. If it is evaluated to be true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason, it is also called the **Entry control loop**
- Once the condition is evaluated to be true, the statements in the loop body are executed. Normally the statements contain an updated value for the variable being processed for the next iteration.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.

Example:

```
// JavaScript code to use while loop
let val = 1;
while (val < 6) {
   console.log(val);
   val += 1;
}
```

Output

```
1
2
3
4
```

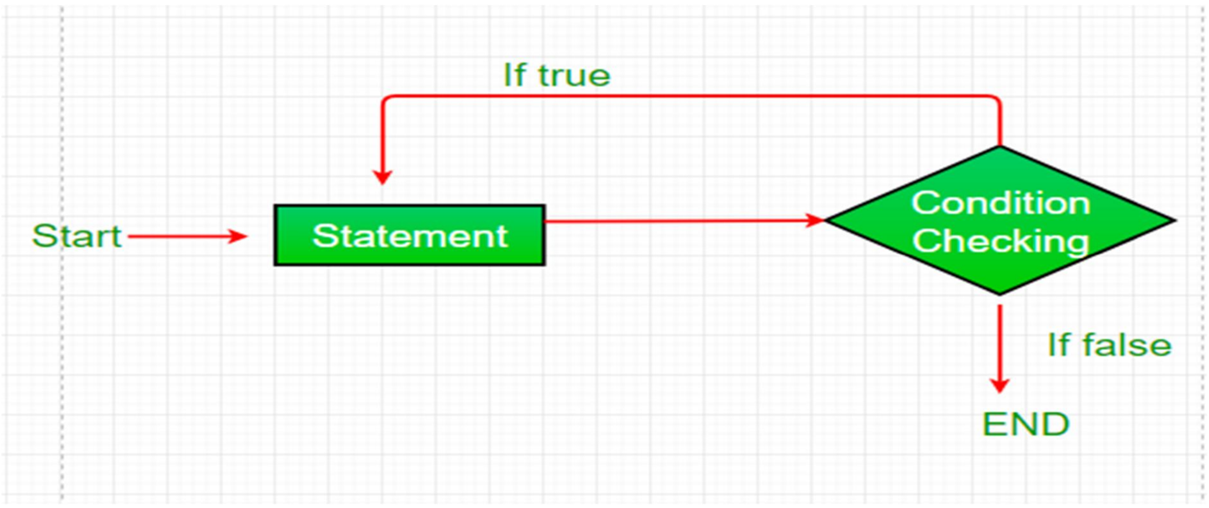
❖ JavaScript do-while Loop

The JS do-while loop is similar to the while loop with the only difference is that it checks for the condition after executing the statements, and therefore is an example of an **Exit Control Loop.** It executes loop content at least once event the condition is false.

Syntax

```
do {
   Statements...
}
while (condition);
```

Flowchart



- The do-while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements and update of the variable value, the condition is checked for a true or false value. If it is evaluated to be true, the next iteration of the loop starts.
- When the condition becomes false, the loop terminates which marks the end of its life cycle.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked and therefore is an example of the exit control loop.

Example:

```
let test = 1;
do {
   console.log(test);
   test++;
} while(test <= 5)
```

Output

```
1
2
3
4
5
```

❖ JavaScript for-in Loop

JS for-in loop is used to iterate over the properties of an object. The for-in loop iterates only over those keys of an object which have their enumerable property set to “true”.

Syntax

```
for(let variable_name in object_name) {
   // Statement
}
```

Example: This example shows the use of for-in loop.

```
let myObj = { x: 1, y: 2, z: 3 };
for (let key in myObj) {
   console.log(key, myObj[key]);
}
```

Output

```
x 1
y 2
z 3
```

❖ JavaScript for-of Loop

JS for-of loop is used to iterate the iterable objects for example – array, object, set and map. It directly iterate the value of the given iterable object and has more concise syntax than for loop.

Syntax:

```
for(let variable_name of  object_name) {
   // Statement
}
```

Example: This example shows the use of for-of loop.

```
let arr = [1, 2, 3, 4, 5];
for (let value of arr) {
   console.log(value);
}
```

Output

```
1
2
3
4
5
```

❖ JavaScript Labeled Statement

JS label keyword does not include a goto keyword. Users can use the continue keyword with the label statement. Furthermore, users can use the break keyword to terminate the loop/block. You can also use the break keyword without defining the label but it terminates only the parent loop/block. To terminate the outer loop from the inner loop using the break keyword, users need to define the label.

Syntax:

```
Label:
   statement (loop or block of code)
```

Example

```
let sum = 0, a = 1;
// Label for outer loop
outerloop: while (true) {
   a = 1;
   // Label for inner loop
   innerloop: while (a < 3) {
      sum += a;
```

```
        if (sum > 12) {
            // Break outer loop from inner loop
            break outerloop;
        }
        console.log("sum = " + sum);
        a++;
    }
}
```

Output

```
sum = 1
sum = 3
sum = 4
sum = 6
sum = 7
sum = 9
sum = 10
sum = 12
```

❖ JavaScript Break Statement

JS break statement is used to terminate the execution of the loop or switch statement when the condition is true.

Syntax

```
break;
```

Example

```
for (let i = 1; i < 6; i++) {
    if (i == 4)
        break;
    console.log(i);
}
```

Output

```
1
2
3
```

❖ JavaScript Continue Statement

JS continue statement is used to break the iteration of the loop and follow with the next iteration. The break in iteration is possible only when the specified condition going to occur. The major difference between the continue and break statement is that the break statement breaks out of the loop completely while continue is used to break one statement and iterate to the next statement.

Syntax

```
continue;
```

Example

```
for (let i = 0; i < 11; i++) {
    if (i % 2 == 0)
        continue;

    console.log(i);
}
```

Output

```
1
3
5
7
9
```

❖ JavaScript Infinite Loop (Loop Error)

One of the most common mistakes while implementing any sort of loop is that it may not ever exit, i.e. the loop runs for infinite times. This happens when the condition fails for some reason.

Example: This example shows an infinite loop.

```
// JavaScript program to illustrate infinite loop
// Infinite loop because condition is not false
// condition should have been i>0.
for (let i = 5; i != 0; i -= 2) {
    console.log(i);
}
let x = 5;
// Infinite loop because update statement
// is not provided
while (x == 5) {
   console.log("In the loop");
}
```

JavaScript switch statement provides a way to execute different code blocks based on different conditions. It’s an alternative to using multiple if...else if...else statements when you have multiple conditions to check.

❖ Switch Statement Syntax

```
switch (expression) {
   case value1:
      // code block 1;
      break;
   case value2:
       // code block 2;
      break;
  ...
   default:
     // default code block;
}
```

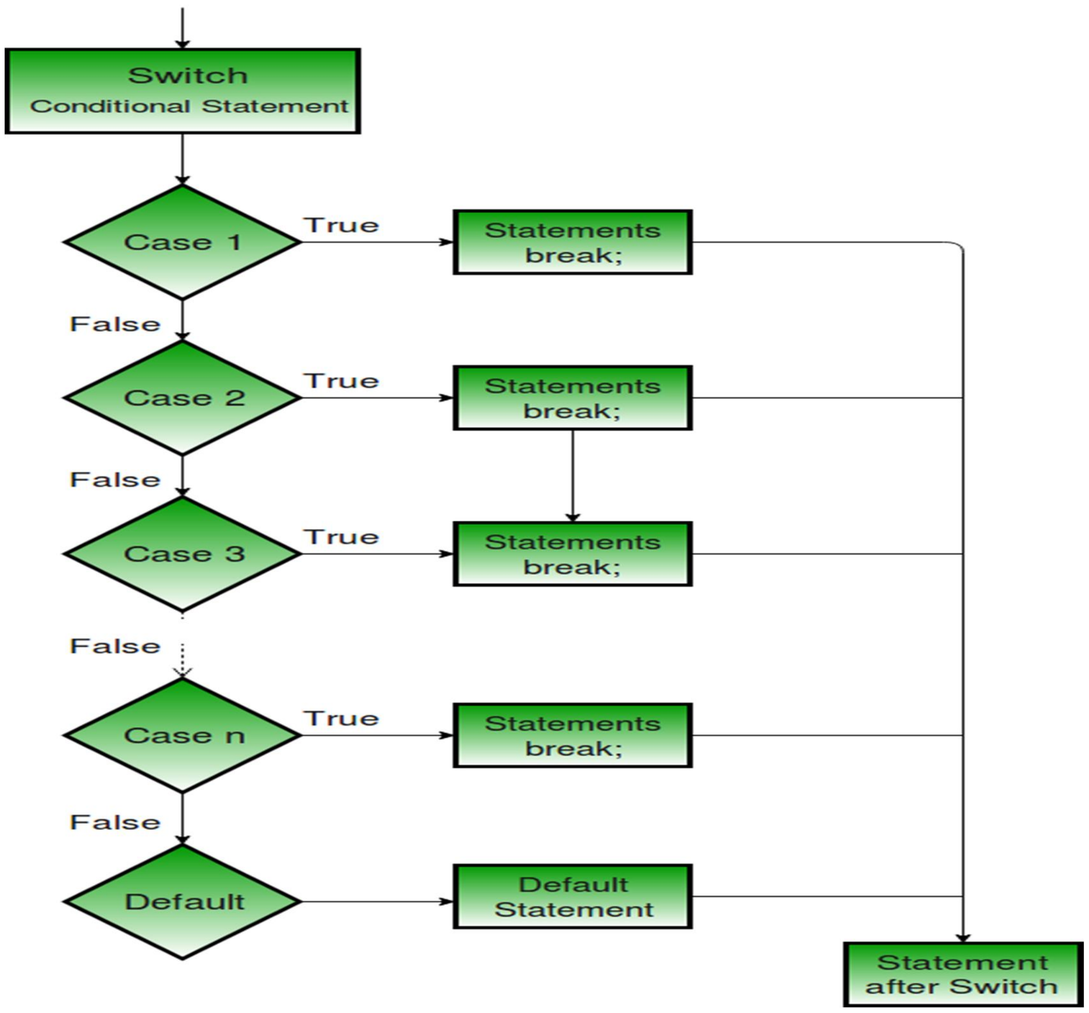
- **Expression** is the value that you want to compare.
- Case value1, case value2, etc., represent the possible values of the expression.
- break statement terminates the switch statement. Without it, execution will continue into the next case.
- Default specifies the code to run if none of the cases match the expression.

❖ How Switch Statement Works

- **Evaluation**: The expression inside the switch statement is evaluated once.
- **Comparison**: The value of the expression is compared with each case label (using strict equality ===).
- **Execution**: If a match is found, the corresponding code block following the matching case label is executed. If no match is found, the execution jumps to the default case (if present) or continues with the next statement after the switch block.

- **Break Statement**: After executing a code block, the break statement terminates the switch statement, preventing execution from falling through to subsequent cases. If break is omitted, execution will continue to the next case (known as "fall-through").
- **Default Case**: The default case is optional. If no match is found, the code block under default is executed.

❖ **Flowchart of Switch Statement**



❖ **Switch Statement Example:**

Here, we will print the day name on day 3.

```
let day = 3;
let dayName;

switch (day) {
   case 1:
      dayName = "Monday";
```

```
        break;
    case 2:
        dayName = "Tuesday";
        break;
    case 3:
        dayName = "Wednesday";
        break;
    case 4:
        dayName = "Thursday";
        break;
    case 5:
        dayName = "Friday";
        break;
    case 6:
        dayName = "Saturday";
        break;
    case 7:
        dayName = "Sunday";
        break;
    default:
        dayName = "Invalid day";
}
console.log(dayName); // Output: Wednesday
```

19.What is the use of is Nan function?

The JavaScript **isNaN()** Function is used to check whether a given value is an illegal number or not. It returns true if the value is a NaN else returns false. It is different from the Number.isNaN() Method.

Syntax:

```
isNaN( value )
```

Parameter Values: This method accepts a single parameter as mentioned above and described below:

value: It is a required value passed in the isNaN() function.

Return Value: It returns a Boolean value i.e. returns true if the value is NaN else returns false.

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

The AND operator

The logical AND operator is used between two operands in an expression like this:

```
operand1 && operand2
```

The operator returns the second operand **if the first operand is a truthy value**. If the first operand is a falsy value, the operator would return the first operand instead. As we have seen, a truthy value is a value that evaluates to true.

The OR operator

The logical OR operator is used between two operands in an expression like this:

```
operand1 || operand2
```

The operator returns the first operand **if the first operand is a truthy value**. If the first operand is a falsy value, the operator would return the second operand instead.

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

The void operator is a valuable tool in JavaScript that evaluates an expression and returns the undefined value. It is frequently used to simply obtain the undefined primitive value by using void(0) or void 0. Even, it's worth noting that the global variable undefined can be used as a substitute for the void operator in most cases, provided it has not been reassigned to a non-default value.

Examples:

```
<a href="JavaScript:void(0)">Click me, nothing will happen</a>
```

This anchor tag specifies a hyperlink with a `javascript:void(0)` URL. When clicked, the JavaScript function void is executed, which returns `undefined` and has no effect on the page. The purpose of using `javascript:void(0)` as the href value is to prevent the page from refreshing and changing the URL when the link is clicked. It's often used when a link is desired but no action is needed to happen.

22. Check Number Is Positive or Negative in JavaScript?

Check program

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

    Check program

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

    Check program

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

    Check program

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

    Check program

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

    Check program

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

    Check program

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

       Check program

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

(Conditional looping logic Question)

30. What are the looping structures in JavaScript? Any one Example?

JavaScript has several looping structures, including:

- **for loop**: used to iterate over a range of values, typically using a counter variable.
- **while loop**: used to repeatedly execute a block of code as long as a specified condition is true.
- **do-while loop**: similar to the while loop, but the code block is executed at least once before the condition is checked.
- **for-in loop**: used to iterate over the properties of an object.
- **for-of loop**: introduced in ES6, used to iterate over iterable objects such as arrays, strings, and maps.

// JavaScript program to illustrate for loop

```
let x;
// for loop begins when x = 2
// and runs till x <= 4
for (x = 2; x <= 4; x++)
{
console.log("Value of x: " + x);
}
```

31. Write a print 972 to 897 using for loop in JS?

Check program

32. Write to print factorial of given number?

Check program

33. Write to print Fibonacci series up to given numbers?

Check program

34. **Write to print number in reverse order e.g.: number = 64728 ---> reverse =82746 in JS?**

    Check program

35. **Write a program make a summation of given number (E.g., 1523 Ans: - 11) in JS?**

    Check program

36. **Write a program you have to make a summation of first and last Digit. (E.g., 1234 Ans: - 5) in JS?**

    Check program

37. **Use console.log() and escape characters to print the following pattern in JS?**

```
1 1 1 1 1
2 1 2 4 8
3 1 3 9 27
4 1 4 16 64
5 1 5 25 125
```

    Check program

38. **Use pattern in console.log in JS?**

```
1)  1
    1 0
    1 0 1
    1 0 1 0
    1 0 1 0 1
```

    Check program

2)
```
A
B C
D E F
G H I J
K L M N O
```

Check program

3)
```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

Check program

4)
```
*
* *
* * *
* * * *
* * * * *
```

Check program

39. Accept 3 numbers from user using while loop and check each numbers palindrome?

Check program

(Array and object Question)

40. **Write a JavaScript Program to display the current day and time in the following format. Sample Output: Today is Friday. Current Time is 12 PM: 12 : 22 2 ?**

    Check program

41. **Write a JavaScript program to get the current date?**

    Check program

42. **Write a JavaScript program to compare two objects?**

    Check program

43. **Write a JavaScript program to convert an array of objects into CSV string?**

    Check program

44. **Write a JavaScript program to capitalize first letter of a string?**

    Check program

45. **Write a JavaScript program to determine if a variable is array?**

    Check program

46. **Write a JavaScript program to clone an array?**

    Check program

47. What is the drawback of declaring methods directly in JavaScript objects?

1. **Memory Inefficiency**: Each object instance gets its own copy of the method. This can lead to increased memory usage, especially if many instances of the object are created, since each instance will have its own copy of the method.

```
function Person(name) {
   this.name = name;
   this.sayHello = function() {
      console.log('Hello, ' + this.name);
   };
}
let person1 = new Person('Alice');
let person2 = new Person('Bob');
// person1 and person2 have separate copies of the sayHello method
```

2. **Difficulty in Sharing and Inheritance**: Methods declared directly in objects cannot be easily shared or inherited. If you need to create a subclass or reuse the method in another object, you might end up duplicating code.

```
let obj1 = {
   greet: function() {
      console.log('Hello');
   }
};
let obj2 = {
   greet: obj1.greet // reference to the same method, but not a true inheritance
};
```

3. **Performance Issues**: In scenarios where objects are created frequently, having methods declared directly can lead to performance issues due to repeated creation and memory allocation for these methods.

4. **Lack of Prototypal Inheritance**: JavaScript's prototypal inheritance system is not utilized when methods are declared directly in objects. Prototypal inheritance allows methods to be defined once on the prototype and shared across all instances of an object.

```
function Person(name) {
   this.name = name;
}
Person.prototype.sayHello = function() {
   console.log('Hello, ' + this.name);
};
let person1 = new Person('Alice');
let person2 = new Person('Bob');
// person1 and person2 share the same sayHello method on the prototype
```

5. **Reduced Maintainability**: As the application grows, maintaining and debugging code with methods declared directly in objects can become cumbersome. It’s easier to manage code with methods on prototypes or within classes, especially when dealing with complex inheritance and method overriding scenarios.

Using prototypes or ES6 classes to declare methods helps to avoid these issues, promoting better memory management, easier inheritance, and overall code maintainability.

48. Print the length of the string on the browser console using console.log()?

Check program

49. Change all the string characters to capital letters using toUpperCase() method?

Check program

50. What is the drawback of declaring methods directly in JavaScript objects?

As it is above Q.N.47

51. Write a JavaScript program to get the current date. Expected Output : mm-dd-yyyy, mm/dd/yyyy or dd-mm-yyyy, dd/mm/yyyy?

Check program

52. Use indexOf to determine the position of the first occurrence of a in 30 Days Of JavaScript?

Check program

53. Use lastIndexOf to determine the position of the last occurrence of a in 30 Days Of JavaScript?

Check program

54. Form Validation in JS?

If a form field (fname) is empty, this function alerts a message, and returns false, to prevent the form from being submitted:

```
function validateForm()
{
        let x = document.forms["myForm"]["fname"].value;
        if (x == "")
        {
        alert("Name must be filled out");
        return false;
        }
}
```

The function can be called when the form is submitted:

HTML Form Example

```
<form name="myForm" action="/action_page.php" onsubmit="return
validateForm()" method="post">
Name: <input type="text" name="fname">
<input type="submit" value="Submit">
</form>
```

55. Form in Email, number, Password, Validation?

Check program

56. Dynamic Form Validation in JS?

Check program

57. How many type of JS Event? How to use it?

JavaScript Events are **actions or occurrences** that happen in the browser. They can be triggered by various user interactions or by the browser itself.
Common events include mouse clicks, keyboard presses, page loads, and form submissions. Event handlers are JavaScript functions that respond to these events, allowing developers to create interactive web applications.
Syntax:

```
        <HTML-element Event-Type = "Action to be performed">
```

❖ Common JavaScript Events

- **onclick**: Triggered when an element is clicked.
- **onmouseover**: Fired when the mouse pointer moves over an element.
- **onmouseout**: Occurs when the mouse pointer leaves an element.
- **onkeydown**: Fired when a key is pressed down.
- **onkeyup**: Fired when a key is released.
- **onchange**: Triggered when the value of an input element changes.
- **onload**: Occurs when a page has finished loading.
- **onsubmit**: Fired when a form is submitted.
- **onfocus**: Occurs when an element gets focus.
- **onblur**: Fired when an element loses focus.

❖ JavaScript Events Examples

Example 1: Here, we will display a message in the alert box when the button is clicked using onClick() event. This HTML document features a button styled to appear in the middle of the page. When clicked, the button triggers the `hiThere()` JavaScript function, which displays an alert box with the message “Hi there!”.

```
<!doctype html>
<html>
<head>
  <script>
    function hiThere() {
        alert('Hi there!');
      }
   </script>
</head>

<body>
   <button type="button"
        onclick="hiThere()"
        style="margin-left: 50%;">
        Click me event
   </button>
</body>
</html>
```

58. What is Bom vs Dom in JS?

Document Object Model (DOM) is a programming interface for HTML and XML documents, that allows to create, manipulate, or delete the element from the document. It defines the logical structure of documents and the way a document is accessed and manipulated. With the help of DOM, the webpage can be represented in a structured hierarchy, i.e., we can easily access and manipulate tags, IDs, classes, Attributes, or Elements of HTML using commands or methods provided by the Document object, that will guide the programmers and users to understand the document in an easier manner.

HTML is used to structure the web pages and Javascript is used to add behavior to our web pages. When an HTML file is loaded into the browser, the javascript can not understand the HTML document directly. So, a corresponding document is created(DOM). DOM is basically the representation of the same HTML document but in a different format with the use of objects. DOM provides several methods to find & manipulate the behavior of the HTML element:

- **getElementById() Method:** This method returns the elements that have given an ID which is passed to the function.
- **getElementsByClassName() Method:** This method in Javascript returns an object containing all the elements with the specified class names in the document as objects: This
- **getElementsByName() Method:** This method returns the collection of all elements of a particular document by name.
- **getElementsByTagName() Method:** This method in HTML returns the collection of all the elements in the document with the given tag name.

- **querySelector() Method:** This method in HTML is used to return the first element that matches a specified CSS selector(s) in the document.
- **querySelectorAll() Method:** This method is used to return a collection of an element's child elements that match a specified CSS selector(s), as a static NodeList object.

Browser Object Model (BOM) is a browser-specific convention referring to all the objects exposed by the web browser. The BOM allows JavaScript to "interact with" the browser. The window object represents a browser window and all its corresponding features. A window object is created automatically by the browser itself. Java Script's window.screen object contains information about the user's screen. It can also be written without the window prefix. This Object Model supports the following Window properties:

- **screen.width:** The screen.width property returns the user's screen width in pixels.
- **screen.height:** The screen.height property returns the user's screen height in pixels.
- **screen.availWidth:** The screen.availWidth property returns the user's screen width in pixels, excluding the interface features.
- **screen.availHeight:** The screen.availHeight property returns the user's screen height in pixels excluding the interface features.
- **screen.colorDepth:** The screen.colorDepth property returns the bits (number) to be used to display one color.
- **screen.pixelDepth:** The screen.pixelDepth property returns the pixel depth of the screen.

The BOM also supports the following Window methods:

- **window.open() Method:** This method is used to open a new tab or window with the specified URL and name.
- **window.close() Method:** This method, is used for closing a certain window or tab of the browser which was previously opened by the *window.open()* method.
- **window.moveTo() Method:** This method is used in the window to move the window from the left and top coordinates.
- **window moveBy() Method:** This method is used for moving a window with a specified number of pixels relative to its current coordinates.
- **window.resizeTo() Method:** This method is used to resize a window to the specified width and height.

Document Object Model (DOM)	*Browser Object Model (BOM)*
It mainly focuses on the structure of the displayed document.	It mainly focuses on browser-specific functionality.
It facilitates a standardized interface to access and modify the elements and content of an HTML or XML document.	It allows JavaScript to interact with browser features beyond the scope of manipulating the document structure.
When an HTML document gets loaded in the browser, then it becomes a document object.	In this case, the window object will be created automatically by the browser.
It facilitates access and manipulation, along with dynamically updating the structure, content, and styling of the web document.	It facilitates the different functionality for governing the browser window, handling the navigation, managing history, and accessing browser-related information.
It provides direct access control to the content of the web document, along with permitting the traversal and modification of its elements and attributes.	It doesn't have any access to the content of the web document directly.

59.Array vs object deferences in JS?

An array is a special variable that can hold more than one value at a time. It is a list-like structure that consists of an ordered collection of elements, each identified by an index or a key. Arrays are declared using [] square brackets notation in JavaScript.

```
let arr = [1, 2, 3, 4, 5]; // declaration of an array
```

On the other hand, an object is a collection of key-value pairs that is used to store various related data. An object is referred to as an instance of its object type or class. Objects consist of properties which have keys and values. Objects are declared using {} curly braces notation in JavaScript.

```
let person = { name: "John", age: 30, occupation: "Developer"}; // declaration of an object
```

The key difference between Arrays and Objects is how the data is organized. Arrays are used when we need to collection a list of elements of the same data type or structure. On the other hand, objects are used when grouping multiple sets of data that belong together using labels, where each property or key has its own value of any type.

Difference between an Array and an Array of objects:

Array	Array of objects
Arrays are best to use when the elements are **numbers**.	Objects are best to use when the elements' **strings (text)**.
The data inside an array is known as **Elements.**	The data inside objects are known as **Properties** which consists of a **key** and a **value.**
The elements can be manipulated using [].	The properties can be manipulated using both**.DOT** notation and [].
The elements can be popped out of an array using the **pop()** function.	The keys or properties can be deleted by using the **delete** keyword.
Iterating through an array is possible using **For loop, For..in, For..of,** and **ForEach().**	Iterating through an array of objects is possible using **For..in, For..of, and ForEach().**

60.Split the string into an array using split() Method?

The split() method splits a string into an array of substrings.
The split() method returns the new array.
The split() method does not change the original string.

If (" ") is used as separator, the string is split between words.

Check program

61. Check if the string contains a word Script using includes() method?

Check program

62. Change all the string characters to lowercase letters using toLowerCase() Method.

Check program

63. What is Character at index 15 in '30 Days of JavaScript' string? Use charAt() method.

Check program

64. Copy to one string to another string in JS?

Check program

65. Find the length of a string without using liabraryFunction?

Check program

66. What is JavaScript?

JavaScript is a *lightweight*, *cross-platform*, *single-threaded*, and *interpreted compiled* programming language. It is also known as the scripting language for webpages. It is well-known for the development of web pages, and many non-browser environments also use it.
JavaScript is a **weakly typed language** **(dynamically typed)**. JavaScript can be used for **Client-side** developments as well as **Server-side** developments. JavaScript is both an imperative and declarative type of language. JavaScript contains a standard library of objects, like **Array**, **Date**, and **Math**, and a core set of language elements like **operators**, **control structures**, and **statements**.

67. What is the use of isNaN function?

The JavaScript **isNaN()** Function is used to check whether a given value is an illegal number or not. It returns true if the value is a NaN else returns false. It is different from the Number.isNaN() Method.

Syntax:

```
isNaN( value )
```

Parameter Values: This method accepts a single parameter as mentioned above and described below:
value: It is a required value passed in the isNaN() function.
Return Value: It returns a Boolean value i.e. returns true if the value is NaN else returns false.

68. What is negative Infinity?

The **negative infinity** in JavaScript is a constant value that is used to represent a value that is the lowest available. This means that no other number is lesser than this value. It can be generated using a self-made function or by an arithmetic operation.

Note: JavaScript shows the NEGATIVE_INFINITY value as -Infinity.

Negative infinity is different from mathematical infinity in the following ways:

- Negative infinity results in **-0**(different from 0) when divided by any other number.
- When divided by itself or positive infinity, negative infinity return NaN
- Negative infinity, when divided by any positive number (apart from positive infinity) is negative infinity.
- Negative infinity, divided by any negative number (apart from negative infinity) is positive infinity.
- If we multiply negative infinity with NaN, we will get NaN as a result.
- The product of 0 and negative infinity is Nan.
- The product of two negative infinities is always a positive infinity.
- The product of both positive and negative infinity is always negative infinity.

Syntax:

```
Number.NEGATIVE_INFINITY
```

69. Which company developed JavaScript?

JavaScript was invented by **Brendan Eich** in 1995.

It was developed for **Netscape 2**, and became the **ECMA-262** standard in 1997.

After Netscape handed JavaScript over to ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser. Mozilla's latest version was 1.8.5. (Identical to ES5).

Internet Explorer (IE4) was the first browser to support ECMA-262 Edition 1 (ES1).

70. What are undeclared and undefined variables?

Undefined: It occurs when a variable has been declared but has not been assigned any value. Undefined is not a keyword.

Undeclared: It occurs when we try to access any variable that is not initialized or declared earlier using the *var* or *const keyword*. If we use *'typeof'* operator to get the value of an undeclared variable, we will face the *runtime error* with the return value as **"undefined"**. The scope of the undeclared variables is always global.

Undefined:

```
let geek;
undefined
console.log(geek)
```

Undeclared:

```
// ReferenceError: myVariable is not defined
  console.log(myVariable)
```

71. Write the code for adding new elements dynamically?

Javascript is a very important language when it comes to learning how the browser works. Often there are times we would like to add dynamic elements/content to our web pages. This post deals with all of that.

Creation of new element: New elements can be created in JS by using the **createElement()** method.

Syntax:

```
document.createElement("<tagName>");
// Where <tagName> can be any HTML
// tagName like div, ul, button, etc.

// newDiv element has been created
For Eg: let newDiv = document.createElement("div");
```

Once the element has been created, let’s move on to the setting of attributes of the newly created element.

Setting the attributes of the created element: Attributes can be set using **setAttribute()** method.
The syntax and example are as follows:

```
Element.setAttribute(name, value);
// Where Element is the name of web element.
// Here, we have created newDiv.
// Where name is the attribute name and
// value is the value that needs to be set

For Eg: newDiv.setAttribute("class","container");
```

```
<!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>
    .box{
      height: 300px;
      width: 300px;
      border: 2px solid;
    }
    .box h1{
      background-color: black;
      color: white;
      }
      .item{
         height: 200px;
         width: 200px;
         border: 2px solid;
         background-color: violet;
      }
   </style>
</head>
<body>
   <div class="box">
      <h1>hello h1</h1>
   </div>
   <script>

      let btn=document.createElement("button");
      btn.innerText="click me";
      btn.style.backgroundColor="grey"
      btn.style.color="white";

      let div=document.querySelector(".box");
      div.before(btn);

      let item=document.createElement("div");
      item.setAttribute("class","item");

      div.append(item);
   </script>
</body>
</html>
```

72. What is the difference between ViewState and SessionState?

View state can only be visible from a single page and not multiple pages.
Session state value availability is across all pages available in a user session.
It will retain values in the event of a postback operation occurring. In session state, user data remains in the server.

ViewState	SessionState
Maintained at page level only.	Maintained at session level.
View state can only be visible from a single page and not multiple pages.	Session state value availability is across all pages available in a user session.
It will retain values in the event of a postback operation occurring.	In session state, user data remains in the server. Data is available to user until the browser is closed or there is session expiration.
Information is stored on the client's end only.	Information is stored on the server.
used to allow the persistence of page-instance-specific data.	used for the persistence of user-specific data on the server's end.
ViewState values are lost/cleared when new page is loaded.	SessionState can be cleared by programmer or user or in case of timeouts.

73. What is === operator?

JavaScript Strict Equality Operator is used to compare two operands and return true if both the value and type of operands are the same. Since type conversion is not done, so even if the value stored in operands is the same but their type is different the operation will return false.

Syntax:

```
a===b
```

74. How can the style/class of an element be changed?

In this article, we will learn how we can change the style/class of an element. If you want to build a cool website or app then UI plays an important role. We can change, add or remove any CSS property from an HTML element on the occurrence of any event with the help of JavaScript. There are two common approaches that allow us to achieve this task.

- style.property
- Changing the class itself

Approach 1: Changing CSS with the help of the style property:
Syntax:

```
document.getElementById("id").style.property = new_style
```

Example: In this example, we have built a PAN number validator. First, we will take the input value and match it with a regex pattern. If it matches then using JavaScript add an inline style on the <p> tag. Otherwise, add a different style on the <p> tag.

```
<!DOCTYPE html>
<html lang="en">
<body>
   <h1 style="color: green;">GeeksforGeeks</h1>
   <h2> How can the style/class of an element be changed?</h2>
   <b>Validate Pan Number</b>
   <input type="text" id="pan" />
   <p></p>
   <button id="submit">Validate</button>
   <script>
      const btn = document.getElementById("submit");
      btn.addEventListener("click", function () {
         const pan = document.getElementById("pan").value;
         const para = document.querySelector("p");

         let regex = /([A-Z]){5}([0-9]){4}([A-Z]){1}$/;
         if (regex.test(pan.toUpperCase())) {
            para.innerHTML = "Hurrey It's correct";

            // Inline style
            para.style.color = "green";
         } else {
            para.innerHTML = "OOps It's wrong!";

            // Inline style
            para.style.color = "red";
         }
      });
   </script>
</body>
</html>
```

Approach 2: Changing the class itself – We can use two properties that can be used to manipulate the classes.
The classList Property: The **classList** is a read-only property that returns the CSS class names of an element as a DOMTokenList object.
Syntax:

```
document.getElementById("id").classList
```

You can use the below-mentioned methods to add classes, remove classes, and toggle between different classes respectively.

- **The add() method:** It adds one or more classes.
- **The remove() method:** It removes one or more classes.
- **The toggle() method:** If the class does not exist it adds it and returns true. It removes the class and returns false. The second boolean argument forces the class to be added or removed.

Example: In this example, we are using the above-explained approach.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <style>
      .hide {
         display: none;
      }
      .blueColor {
         color: blue;
      }
   </style>
</head>
<body>
   <h1 style="color: green;">GeeksforGeeks</h1>
   <h2>How can the style/class of an element be changed?</h2>
   <h3>Hide and Show the Para</h3>
   <p>
      GeeksforGeeks is a computer science portal for geeks. This platform has been
      designed for every geek wishing to expand their knowledge, share their
      knowledge,and is ready to grab their dream job. GFG have millions of articles, live
      as well as online courses, thousands of tutorials and much more just for the geek
      inside you.
   </p>
   <button id="hide">Hide</button>
   <button id="show">Show</button>
   <button id="color">Change Color</button>
   <script>
      const btn_hide = document.getElementById("hide");
      const btn_show = document.getElementById("show");
      const btn_color = document.getElementById("color");
      const para = document.querySelector("p");
      btn_hide.addEventListener("click", function () {
         para.classList.add("hide");
      });

      btn_show.addEventListener("click", function () {
         para.classList.remove("hide");
      });

      btn_color.addEventListener("click", function () {
         para.classList.toggle("blueColor");
      });
   </script>
</body>
</html>
```

In the above example, we define two manipulation classes “hide” and “toggleColor” which hide an element and change color to blue respectively. When the Hide button is clicked, the hide class is added to the “p” tag which hides it. When the Show button is clicked, it removes the present hide class from the “p” tag. When the Change Color button is clicked once it adds “toggleColor” class to the p tag(which makes the text color blue) and when clicked again it removes the toggleColor class.

The className Property: This property is used to set the current class of the element to the specified class.
Syntax:

```
document.getElementById("id").className = class
```

Example: In this example, we are using the className property to change the color of the element.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <style>
      .colorBlue {
         color: blue;
      }
      .colorRed {
         color: red;
      }
   </style>
</head>
<body>
   <h1 style="color: green;"> GeeksforGeek</h1>
   <h2>How can the style/class of an element be changed?</h2>
   <h3>className Example</h3>
   <p class="colorBlue">
GeeksforGeeks is a computer science portal for geeks.This platform has been designed
for every geek wishing to expand their knowledge, share their knowledge and is
ready to grab their dream job. GFG have millions of articles, live as well as online
courses, thousands of tutorials and much more just for the geek inside you.
   </p>

   <button id="submit">Change Color</button>
```

```
    <script>
        const btn = document.getElementById("submit");
        const para = document.querySelector("p");

        btn.addEventListener("click", function () {
            para.className = "colorRed";
        });
    </script>
</body>
</html>
```

75.How to read and write a file using JavaScript?

readFile() and rs. writeFile() methods are used to read and write of a file using javascript. The file is read using the fs. readFile() function, which is an inbuilt method.

Handling file operations such as reading and writing is a common requirement in many JavaScript applications, especially on the server side using Node.js. This article explains how to perform these operations using Node.js's built-in fs (File System) module.

The fs Module
The fs module provides an API for interacting with the file system in a manner closely modeled around standard POSIX functions. It can be used to perform operations such as reading, writing, updating, and deleting files.
The fs.readFile() and rs.writeFile() methods are used to read and write of a file using javascript.

fs.readFile()
The file is read using the fs.readFile() function, which is an inbuilt method. This technique reads the full file into memory and stores it in a buffer.
Syntax:

```
fs.readFile( file_name, encoding, callback_function )
```

Parameters:

- **filename:** It contains the filename to be read, or the whole path if the file is saved elsewhere.
- **encoding:** It stores the file's encoding. 'utf8' is the default setting.

- **callback function:** This is a function that is invoked after the file has been read. It requires two inputs:
- **err:** If there was an error.
- **data:** The file's content.
- **Return Value:** It returns the contents contained in the file, as well as any errors that may have occurred.

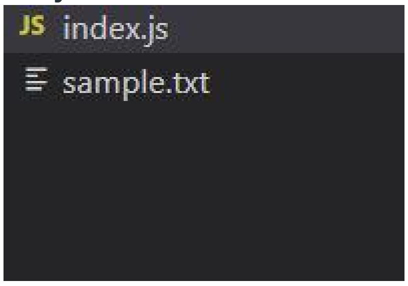
fs.writeFile()
The fs.writeFile() function is used to write data to a file in an asynchronous manner. If the file already exists, it will be replaced.
Syntax:

```
fs.writeFile( file_name, data, options, callback )
```

Parameters:

- **file_name**: It's a string, a buffer, a URL, or a file description integer that specifies the location of the file to be written. When you use a file descriptor, it will function similarly to the fs. write() method.
- **data**: The data that will be sent to the file is a string, Buffer, TypedArray, or DataView.
- **options:** It's a string or object that may be used to indicate optional output options. It includes three more parameters that may be selected.
- **encoding**: It's a string value that indicates the file's encoding. 'utf8' is the default setting.
- **mode**: The file mode is specified by an integer number called mode. 0o666 is the default value.
- **flag**: This is a string that indicates the file-writing flag. 'w' is the default value.
- **callback**: This function gets invoked when the method is run.
- **err**: If the process fails, this is the error that will be thrown.

Let's understand how to write and read files using an example, In the below example, we are creating a file using the writeFile() method and reading the content of the file readFile() method.
Project Structure:



```
index.js
sample.txt
```

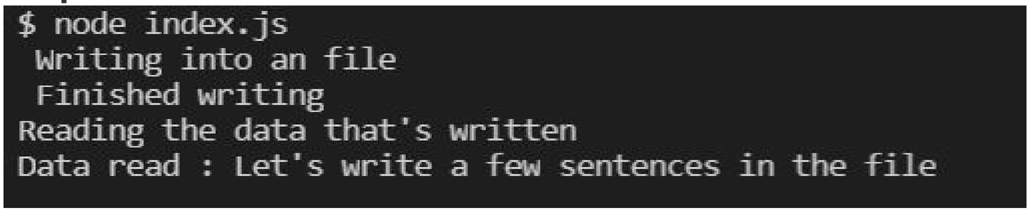
Example: Implementation to Read and Write of a file using JavaScript.
index.js

```
const fs = require("fs");
console.log(" Writing into an file ");
// Sample.txt is an empty file
fs.writeFile(
  "sample.txt",
  "Let's write a few sentences in the file",
  function (err) {
    if (err) {
      return console.error(err);
    }
    // If no error the remaining code executes
    console.log(" Finished writing ");
    console.log("Reading the data that's written");

    // Reading the file
    fs.readFile("sample.txt", function (err, data) {
      if (err) {
        return console.error(err);
      }
      console.log("Data read : " + data.toString());

    });
  }
);
```

Output:



```
$ node index.js
 Writing into an file
 Finished writing
Reading the data that's written
Data read : Let's write a few sentences in the file
```

76.What are all the looping structures in JavaScript?

JavaScript for Loop

A **for loop** is used when you know how many times you need to repeat a certain block of code. It takes three statements.

- Initialization statement
- Condition Statement
- Increment statement

JavaScript while Loop
A **while loop** is used when you don’t know how many times you need to repeat a block of code, but you know the condition that will end the loop.

JavaScript do-while Loop
A **do-while loop** is similar to a while loop, but the block of code is executed at least once, even if the condition is false.

JavaScript for-in Loop
A **for-in loop** is used to loop through the properties of an object.

JavaScript for-of Loop
A **for-of loop** is used to loop through the values of an iterable object (such as an array.

JavaScript forEach loop
A **forEach loop** is a method on arrays that executes a function for each element in the array.

JavaScript map Loop
A **map loop** is a method on arrays that creates a new array by executing a function on each element in the array.

77. How can you convert the string of any base to an integer in JavaScript?

Approach 1: Convert String to an Integer using parseInt() Method
The parseInt() method accepts the string and radix parameter and converts it into an integer.

Syntax:

```
parseInt( Value, radix )
```

Example: In this case, we use the JavaScript parseInt() method to convert a string into an integer.

```
let a = "100";

console.log("Type of a before conversion: " + typeof a);
console.log("Type of a after conversion: " + typeof parseInt(a));
```

Output

```
Type of a before conversion: string
Type of a after conversion: number
```

Explanation
The code initializes a variable a with the string “100”, prints its type before and after converting it to an integer using parseInt()

Approach 2: Convert String to an Integer using Number() Method

The number() method is used to convert primitive data type to a number, if it is not convertible it returns NAN.
Syntax:

```
Number( value )
```

Example: Here, we are using JavaScript Number() method to convert string to an integer.

```
let age = "23";
let name = "Manya";

console.log(
   "Type of name and age before conversion: ",
   typeof age, typeof name
);

console.log(
   "Type of name and age after conversion: ",
   typeof Number(age), typeof Number(name)
);
```

Output

```
Type of name and age before conversion:  string string
Type of name and age after conversion:  number number
```

Explanation
The code initializes variables age and name with string values representing age and a name respectively. It then displays their types before and after conversion to numbers using Number() .

Approach 3: Convert String to an Integer using Unary Operator

The **Unary operator(+)** is used to convert a string, boolean, and non-string to a number.
Syntax:

```
+operand;
```

Example:

The unary operator to convert a string into an number.

```
let age = "23";
let name = "Manya";
const number = '100';

console.log(
   "Type of name, age and number before conversion: ",
   typeof age, typeof name, typeof number
);

console.log(
   "Type of name, age and number after conversion: ",
   typeof +age, typeof +name, typeof +number
);
```

Output

```
Type of name, age and number before conversion:  string string string
Type of name, age and number after conversion:  number number number
```

Explanation

The code initializes variables age , name , and number with string values representing age, a name, and a constant number respectively. It then displays their types before and after conversion to numbers using the unary plus operator + .

Approach 4: Convert String to an Integer using Math.floor() Method

The Math.floor() method is used to convert a string into number.

Syntax:

```
Math.floor( numStr )
```

Example:

The Math.floor() method is used to convert a string into an number.

```
let numStr = "101.45";

let numInt = Math.floor(parseFloat(numStr));
console.log("Type of Number Before Conversion: " + typeof numStr);

console.log("Type of Number After Conversion: " + typeof numInt);
```

Output

```
Type of Number Before Conversion: string
Type of Number After Conversion: number
```

Explanation

Here, parseFloat() is used to convert the string numStr to a floating-point number before applying Math.floor() . This ensures that the result is a number.

Approach 5: Convert String to an Integer using Bitwise Operator

Using bitwise operators like **| 0** or **<< 0** can also perform string-to-integer conversion in a concise manner.

Syntax:

```
numStr | 0;   // or
numStr << 0;
```

Example: The Bitwise Operator can also be used to convert a string into an number.

```
let numStr = "101.45";

let numInt = numStr | 0;

console.log("Type of Number Before Conversion: "
   + typeof numStr);

console.log("Type of Number After Conversion: "
   + typeof parseInt(numInt));
```

Output

```
Type of Number Before Conversion: string
Type of Number After Conversion: number
```

Approach 6: Convert String to an Integer using BigInt() Constructor

The BigInt() constructor can be used to convert a string into a BigInt integer. This is particularly useful when dealing with very large integers that exceed the maximum safe integer limit in JavaScript.

Example:

```
// Example of using BigInt() constructor to convert a string into a BigInt integer
let numStr = "12345678901234567890";

let numBigInt = BigInt(numStr);
```

```
console.log("Type of Number Before Conversion: " + typeof numStr);
console.log("Type of Number After Conversion: " + typeof numBigInt);
console.log(numBigInt);
// Nikunj Sonigara
```

Output

```
Type of Number Before Conversion: string
Type of Number After Conversion: bigint
1234567890123456789 0n
```

Explanation

The code initializes a variable numStr with the string "101.45". It then uses a bitwise OR operation (|) to implicitly convert numStr to an integer, truncating its decimal part. Finally, it displays the type of numInt before and after conversion.

78. What is the function of the delete operator?

The **delete operator** in JavaScript is used to remove a property from an object. It works for both properties owned by the object and those inherited from prototypes. When used on an array item, it creates a 'hole' in the array.

Syntax:

```
delete object
// or
delete object.property
// or
delete object['property']
```

Parameter: It does not take any parameters.

Return type: This operator returns *true* if it removes a property. While deleting an object property that doesn't exist will return a *true* but it will not affect the object. However, while trying to delete a variable or a function will return a *false*.

1. Deleting Object Properties:

- The primary use of `delete` is to remove properties from objects.
- After deletion, accessing the property will return `undefined`.

Ex:

```
let car = {
  brand: 'Toyota',
  model: 'Camry',
  year: 2020
};
delete car.model;
console.log(car); // Output: { brand: 'Toyota', year: 2020 }
console.log(car.model); // Output: undefined
```

2. Behavior with Arrays:

- `delete` can also be used to remove elements from an array, but this leaves a `undefined` or empty slot in the array. The length of the array does not change.

Ex:

```
let numbers = [1, 2, 3, 4];
delete numbers[2];
console.log(numbers); // Output: [1, 2, empty, 4]
console.log(numbers.length); // Output: 4
```

3. Non-Configurable Properties:

- Properties that are marked as non-configurable cannot be deleted. If you attempt to delete a non-configurable property, `delete` will return `false` (and in strict mode, it will throw an error).

Ex:

```
var obj = {};
Object.defineProperty(obj, 'nonConfigurable', {
  value: 'cannot delete this',
  configurable: false
});
delete obj.nonConfigurable; // false in non-strict mode, error in strict mode
console.log(obj.nonConfigurable); // Output: 'cannot delete this'
```

4. Variables Declared with `var`, `let`, or `const`:

- `delete` cannot be used to delete variables declared with `var`, `let`, or `const`. Attempting to do so will return `false` or throw an error in strict mode.

Ex:

```
var x = 10;
delete x; // false
console.log(x); // Output: 10
```

5. Global Object Properties:

- If you declare a variable without `var`, `let`, or `const`, it becomes a property of the global object and can be deleted.

Ex:

```
y = 20; // Becomes a property of the global object
delete y; // true
console.log(typeof y); // Output: 'undefined'
```

In summary, the `delete` operator is used to manage the properties of objects by removing them, and its behavior varies slightly depending on the context in which it's used.

79. What are all the types of Pop up boxes available in JavaScript?

In Javascript, popup boxes are used to display the message or notification to the user. There are three types of pop-up boxes in JavaScript:

- Alert Box
- Prompt Box
- Confirm Box

Alert Box

It is used when a warning message is needed to be produced. When the alert box is displayed to the user, the user needs to press ok and proceed.

Syntax:

```
alert("your Alert here");
```

Prompt Box

It is a type of pop up box which is used to get the user input for further use. After entering the required details user have to click ok to proceed next stage else by pressing the cancel button user returns the null value.

Syntax:

```
prompt("your Prompt here");
```

Confirm Box

It is a type of pop-up box that is used to get authorization or permission from the user. The user has to press the ok or cancel button to proceed.

Syntax:

```
confirm("your query here");
```

80. What is the use of Void (0)?

You might have occasionally came across “javascript:void(0)” in an HTML Document. It is often used when inserting an expression in a web page might produce some unwanted effect. To remove this effect, *“javascript:void(0)”* is used. This expression returns undefined primitive value. This is often used with hyperlinks. Sometimes, you will decide to call some JavaScript from inside a link. Normally, when you click a link, the browser loads a brand new page or refreshes the same page (depending on the URL specified). But you most likely don’t desire this to happen if you have hooked up some JavaScript thereto link. To prevent the page from refreshing, you could use void(0).

Using “#” in anchor tag: When writing the following code in the editor, the web page is refreshed after the alert message is shown.

Example:

```
<h1 style="color:green">GeeksforGeeks</h1>
<h3>without JavaScript:void(0)</h3>
<a href="#" ondblclick="alert('Welcome to Geeks for Geeks')">
   Double click on me </a>
<a href="#" ondblclick="geeks()">
   Double click on me
</a>
<script>
   function geeks(){
   document.getElementById("gfg")
      .innerHTML='Welcome to GeeksforGeeks';
   }
</script>
<p id="gfg"></p>
```

Using “javascript:void(0);” in anchor tag: Writing “javascript:void(0);” in anchor tag can prevent the page to reload and JavaScript functions can be called on single or double clicks easily.

Example:

```
<h1 style="color:green">GeeksforGeeks</h1>
<h3>JavaScript:void(0)</h3>
<a href="javascript:void(0);" ondblclick="geek()">
   Double click on me </a>
<p id="gfg">
```

```
</p>
<script>
   function geek() {
      document.getElementById("gfg").innerHTML = "Welcome to GeeksforGeeks";
   }
</script>
```

81. How can a page be forced to load another page in JavaScript?

In JavaScript, you can force a page to load another page by using the window.location object. There are a few methods to achieve this. To force a page to load another page in JavaScript, we have multiple approaches:
Below are the approaches used to force a page to load another page in JavaScript:

- Using window.location.replace
- Using window.location.assign Property

Approach 1: Using window.location.replace

The **replace** function is used to navigate to a new URL without adding a new record to the history.
Syntax:

```
// Redirect to another page using replace
window.location.replace('https://www.example.com');
```

Example: This example shows the use of the window.location.replace which is used to load another page in JavaScript.

```
<!DOCTYPE html>
<html>
<body>
   <button onclick="replaceLocation()">
      Replace current webpage
   </button>
   <script>
      function replaceLocation() {
         // Replace the current location
         // with new location
```

```
            let newloc = "https://www.geeksforgeeks.org/";
            window.location.replace(newloc);
        }
    </script>
</body>
</html>
```

Approach 2: Using window.location.assign Property

- The **assign** function is similar to the href property as it is also used to navigate to a new URL.
- The assign method, however, does not show the current location, it is only used to go to a new location.
- Unlike the replace method, the assign method adds a new record to history (so that when the user clicks the "Back" button, he/she can return to the current page).

Syntax:

```
window.location.assign("https://geeksforgeeks.org/")
```

Example: This example shows the use of the window.location.assign property

```
<!DOCTYPE html>
<html>
<body>
    <button onclick="assignLocation()">
        Go to new webpage
    </button>
    <script>
        function assignLocation() {
            // Go to another webpage (geeksforgeeks)
            let newloc = "https://www.geeksforgeeks.org/";
            window.location.assign(newloc);
        }
    </script>
</body>
</html>
```

82. What are the disadvantages of using innerHTML in JavaScript?

The innerHTML property is a part of the Document Object Model (DOM) that is used to set or return the HTML content of an element. Where the return value represents the text content of the HTML element. It allows JavaScript code to manipulate a website being displayed. More specifically, it sets or returns the HTML content (the inner HTML) of an element. The innerHTML property is widely used to modify the contents of a webpage as it is the easiest way of modifying DOM. But there are some disadvantages to using innerHTML in JavaScript.

Disadvantages of using innerHTML property in JavaScript:

- **The use of innerHTML very slow:** The process of using innerHTML is much slower as its contents as slowly built, also already parsed contents and elements are also re-parsed which takes time.
- **Preserves event handlers attached to any DOM elements:** The event handlers do not get attached to the new elements created by setting innerHTML automatically. To do so one has to keep track of the event handlers and attach it to new elements manually. This may cause a memory leak on some browsers.
- **Content is replaced everywhere:** Either you add, append, delete or modify contents on a webpage using innerHTML, all contents is replaced, also all the DOM nodes inside that element are reparsed and recreated.
- **Appending to innerHTML is not supported:** Usually, += is used for appending in JavaScript. But on appending to an Html tag using innerHTML, the whole tag is re-parsed.

Example:

```
<p id="geek">Geeks</p>
title = document.getElementById('#geek')


// The whole "geek" tag is reparsed
title.innerHTML += '<p> forGeeks </p>'
```

- **Old content replaced issue:** The old content is replaced even if object.innerHTML = object.innerHTML + ‘html’ is used instead of object.innerHTML += ‘html’. There is no way of appending without reparsing the whole innerHTML. Therefore, working with innerHTML becomes very slow. String concatenation just does not scale when dynamic DOM elements need to be created as the plus’ and quote openings and closings becomes difficult to track.

- **Can break the document:** There is no proper validation provided by innerHTML, so any valid HTML code can be used. This may break the document of JavaScript. Even broken HTML can be used, which may lead to unexpected problems.
- **Can also be used for Cross-site Scripting(XSS):** The fact that innerHTML can add text and elements to the webpage, can easily be used by malicious users to manipulate and display undesirable or harmful elements within other HTML element tags. Cross-site Scripting may also lead to loss, leak and change of sensitive information.

Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>
        Using innerHTML in JavaScript
    </title>
</head>
  <body style="text-align: center">
    <h1 style="color:green"> GeeksforGeeks  </h1>
    <p id="P">  A computer science portal for geeks. </p>
    <button onclick="geek()"> Try it </button>
    <p id="p"></p>
    <script>
        function geek() {
            var x = document.getElementById("P")
                        .innerHTML;

            document.getElementById("p")
                        .innerHTML = x;

            document.getElementById("p")
                        .style.color = "green";
        }
    </script>
</body>
</html>
```