Ques 1) What is JavaScript. How to use it?

Ans 1) JavaScript is a scripting language used to develop web pages. Developed in Netscape, JS allows developers to create a dynamic and interactive web page to interact with visitors and execute complex actions.

```
<script type="text/JavaScript">

    document. Write ("JavaScript is a simple language for java point learners");

</script>
```

Ques 2) How many type of Variable in JavaScript?

Ans 2) There are two types of variables in JavaScript :

- local variable
- global variable.

Ques 3) Define a Data Types in js?

Ans 3) JavaScript includes primitive and non-primitive data types.

The primitive data types in JavaScript include string, number, boolean, undefined, null, and symbol.

The non-primitive data type includes the object.

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

Ans 4)

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

- Here mul function accept the first argument and return anonymous function which take the second parameter and return anonymous function which take the third parameter and return multiplication of arguments which is being passed in successive.

- In JavaScript function defined inside has access to outer function variable and function is the first class object so it can be returned by function as well and passed as argument in another function.

  The classifications such as:

- A function is an instance of the Object type.
- A function can have properties and has a link back to its constructor method.
- Function can be stored as variable.
- Function can be pass as a parameter to another function.
- Function can be returned from function.

Ques 5) what the deference between undefined and undeclare in JavaScript?

Ans 5)

Undefined	Undeclare
It occurs when a variable has been declared but has not been assigned any value. Undefined is not a keyword.	It occurs when we try to access any variable that is not initialized or declared earlier using the *var* or *const keyword*.
The variables which are written in the code but haven't been assigned any value yet are called undefined.	The variables which don't exist in the memory heap area, ie., not written inside the code, are called undeclared.

Ques 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 6)

```
<script>
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.");

console.log("Spread love everywhere you go. Let no one ever come to you without leaving happier. - Mother
Teresa");

</script>
```

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

Ans 7) If the type of '10' is exactly equal to 10:

```
<script>

console.log(typeof '10' === typeof 10); // false

</script>
```

As you can see, the type of '10' (which is a string) is not exactly equal to the type of 10.

To make them exactly equal, we can convert the string '10' to a number. Here's how you can do it:

```
<script>

let stringNum = '10';
let numberNum = parseInt(stringNum)
console.log(typeof stringNum); // string
console.log(typeof numberNum); // number
console.log(numberNum == 10); // true

</script>
```

By using parseInt(), we converted the string '10' to a number, making them exactly equal in type and value.

Ques 8) write a JavaScript Program to find the area of a triangle?

Ans 8)

```
<body>
    <script>
    const basicValue = prompt('Enter the base of a triangle: ');
    const heightValue = prompt('Enter the height of a triangle: ');

// calculate the area //
    const areaValue = (basicValue * heightValue) / 2;

    console.log(
    `The area of the triangle is ${areaValue}`
    );
    </script>
```

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

Ans 9)

```
    <script>
    function daysUntilChristmas() {
    var today = new Date();
    var currentYear = today.getFullYear();
    var christmas = new Date(currentYear, 11, 25);
    if (today > christmas) {
        christmas.setFullYear(currentYear + 1);
    }
    var difference = christmas.getTime() - today.getTime();
    var daysLeft = Math.ceil(difference / (1000 * 60 * 60 * 24));
    return daysLeft;
}
    var daysLeftUntilChristmas = daysUntilChristmas();
    console.log("Days left until Christmas:", daysLeftUntilChristmas);
    </script>
```

Ques 10) what is Condition Statement?

Ans 10) It allow you to execute specific blocks of code based on conditions. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition.

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

Ans 11) The formula provided, C=4 × a, calculates the circumference of a rectangle. However, this formula is incorrect for finding the circumference of a rectangle. For a rectangle, the formula for finding the circumference (perimeter) is given by:

- C = 2×(length + width) C = 2 \times (length + width) C= 2×(length + width)

```
<script>

function circumferenceOfRectangle(length, width)
{
let a = length + width;
let C = 4 * a;
return C;
}

let length = 5;
let width = 3;
let circumference = circumferenceOfRectangle(length, width);
console.log("Circumference of the rectangle:", circumference);

</script>
```

Ques 12) WAP to convert years into days and days into years?

Ans 12)

```
<script>

// Function to convert years into days
function yearsToDays(years) {
return years * 365; // Assuming a year has 365 days
}

// Function to convert days into years
function daysToYears(days) {
return days / 365; // Assuming a year has 365 days
}

// Example usage:
let years = 3;
let days = 1095; // 3 years * 365 days/year = 1095 days

console.log(years + " years is equal to " + yearsToDays(years) + " days.");
console.log(days + " days is equal to " + daysToYears(days) + " years.");

</script>
```

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

Ans 13) To convert temperature from Fahrenheit to Celsius, you can use the following formula:

- $C=59×(F−32)C = \frac{5}{9} \times (F - 32)C=95×(F−32)$

Where:

- CCC is the temperature in Celsius,
- FFF is the temperature in Fahrenheit.

```
<script>

function fahrenheitToCelsius(fahrenheit) {
let celsius = (5/9) * (fahrenheit - 32);
return celsius;
}

// Example usage:
let fahrenheitTemp = 68; // Example Fahrenheit temperature
let celsiusTemp = fahrenheitToCelsius(fahrenheitTemp);
console.log(fahrenheitTemp + "°F is equal to " + celsiusTemp.toFixed(2) + "°C.");

</script>
```

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

Ans 14)

```
<script>
function getFileExtension(filename) {
// Split the filename by '.' to separate the name and the extension
let parts = filename.split('.');

// Get the last part which represents the extension
let extension = parts[parts.length - 1];

return extension;
}

// Example usage:
let filename = "example_script.js";
let extension = getFileExtension(filename);
console.log("The extension of the file '" + filename + "' is: " + extension);

</script>
```

In this code:

- The get File Extension function takes a filename as input.

- It splits the filename by '.' using the split method, which separates the filename into an array of parts.
- It then retrieves the last part of the array, which represents the extension.
- Finally, it returns the extension.
- The example usage demonstrates how to use this function with an example filename and logs the extension to the console.

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

Ans 15) To evaluate the expression **(5 > 3 && 2 < 4)** in VS Code, we need to understand how it will be handled in the context of a specific programming language. Assuming we are working with JavaScript, which is commonly used in VS Code, the evaluation would proceed as follows:

- Evaluate 5 > 3:

  - This comparison checks if 5 is greater than 3.
  - 5 > 3 is true.

- Evaluate 2 < 4:

  - This comparison checks if 2 is less than 4.
  - 2 < 4 is true.

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

Ans 16) In Visual Studio Code (VS Code), if you evaluate the expression (true && 1 && "hello") in a JavaScript file or the integrated terminal, the result will be "hello".

Here's a breakdown of why this happens:

1. True: This is a truth value.
2. This is also a truth value.
3. "Hello": This is a string, which is truth as well.

The logical AND operator (&&) in JavaScript returns the first false value it encounters, or if all values are truth, it returns the last value. Since all values in this expression are truth, the result will be the last value: "hello".

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

Ans 17) To determine the result of the expression true && false || false && true in JavaScript, as you might evaluate it in VS Code, we need to follow the order of operations for logical operators, which prioritize the logical AND (&&) operator over the logical OR (||) operator.

Here is the step-by-step evaluation:

1. Evaluate true && false:
   - True && false evaluates to false because both operands must be true for the result to be true.

2. Evaluate false && true:
   - False && true evaluates to false because both operands must be true for the result to be true.

3. Evaluate the OR (||) operation:
   - Now, we have false || false.
   - False || false evaluates to false because at least one operand must be true for the result to be true.

So, the result of the expression true && false || false && true in VS Code or any JavaScript environment is false.

Ques 18) what is a Loop and Switch Case in JavaScript define that?

Ans 18) A loop is a programming construct that allows you to repeat a block of code multiple times. JavaScript supports several types of loops:

- for Loop:
  - The for loop repeats a block of code as long as a specified condition is true. It consists of three parts: initialization, condition, and increment/decrement.

```
<script>
for (let i = 0; i < 5; i++) {
console.log(i);  // Outputs 0, 1, 2, 3, 4
}
</script>
```

❖ while Loop:

➢ The while loop executes a block of code as long as the specified condition is true.

```
<script>
let i = 0;
while (i < 5) {
console.log(i);  // Outputs 0, 1, 2, 3, 4
i++;
}
</script>
```

❖ do...while Loop:

➢ The do...while loop is similar to the while loop, but it guarantees that the code block will be executed at least once.

```
<script>

let i = 0;
do {
console.log(i);  // Outputs 0, 1, 2, 3, 4
i++;
} while (i < 5);

</script>
```

❖ for...in Loop:

➢ The for...in loop iterates over the properties of an object.

```
<script>

const obj = {a: 1, b: 2, c: 3};
for (let key in obj) {
console.log(key, obj[key]);  // Outputs "a 1", "b 2", "c 3"
}

</script>
```

❖ for...of Loop:

➢ The for...of loop iterates over alterable objects like arrays, strings, maps, etc.

```
<script>

const array = [10, 20, 30];
for (let value of array) {
console.log(value);  // Outputs 10, 20, 30
}

</script>
```

Switch Case in JavaScript

- The switch statement is used to perform different actions based on different conditions. It's a control statement that executes one block of code among many.

```
<script>
let fruit = 'apple';

switch (fruit) {
case 'banana':
    console.log('Banana is yellow');
    break;
case 'apple':
    console.log('Apple is red');
    break;
case 'grape':
    console.log('Grape is purple');
    break;
default:
    console.log('Unknown fruit');
}
</script>
```

Ques 19) what is the use of is Nan function?

Ans 19) 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.

Here's a basic example of how to use **isNaN** in JavaScript:

```
<script>
let value = NaN;
console.log(isNaN(value));  // Output: true

let number = 42;
console.log(isNaN(number));  // Output: false

let stringValue = "hello";
console.log(isNaN(stringValue));  // Output: true

</script>
```

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

Ans 20)

Objective	Logic AND (&&)	Logic OR (||)
Operation	The **&&** operator evaluates to true if both operands are true; otherwise, it evaluates to false.	The || operator evaluates to true if at least one of the operands is true; otherwise, it evaluates to false.
Short – Circuit Evaluation	If the first operand evaluates to false, the second operand is not evaluated because the result is already determined to be false.	If the first operand evaluates to true, the second operand is not evaluated because the result is already determined to be true.
Usage	• Commonly used in conditions where multiple criteria need to be met. • Can be used to execute code only if multiple conditions are true.	• Commonly used in conditions where at least one of multiple criteria needs to be met. • Can be used to provide default values.

Ques 21) What is the use of void (0)?

Ans 21) In programming, particularly in JavaScript, void(0) is often used to prevent a hyperlink from navigating to a new page or performing any action. Here's a detailed explanation of its use:

- Preventing Default Action in Hyperlinks: When you have a hyperlink (<a> tag) and you want to use it to trigger JavaScript code instead of navigating to a new URL, you can use href="javascript:void(0)". This effectively makes the link do nothing when clicked.

- Stopping Default Behavior: In some cases, you might want to prevent the default behavior of a link or form submission while still allowing

JavaScript to execute. Using void(0) ensures that the browser doesn't navigate away or reload the page.

❖ Example usuage :

- Preventing Navigation

```
<script>

<a href="javascript:void(0)" onclick="alert('Link clicked!');">Click me</a>

</script>
```

In this example, clicking the link will trigger an alert box, but the browser will not navigate to any new page because of the void(0).

- Within an Event Handler

```
<script>

<a href="#" onclick="doSomething(); return false;">Click me</a>

</script>
```

Here, return false; in the on click handler also prevents the default action of the link (similar to void (0)). However, void(0) is sometimes preferred for clarity in certain situations.

Ques 22) Check Number Is Positive or Negative in JavaScript?
Ans 22) In JavaScript, you can check whether a number is positive or negative by using simple conditional statements. Here is a methods to achieve this:

Method: Using if-else Statements

```
<script>

function checkNumber(num) {
if (num > 0) {
    console.log(num + " is positive.");
} else if (num < 0) {
    console.log(num + " is negative.");
} else {
    console.log(num + " is zero.");
}
}

// Example usage:
checkNumber(10); // Output: 10 is positive.
checkNumber(-5); // Output: -5 is negative.
checkNumber(0); // Output: 0 is zero.

</script>
```

Explanation:

- Method: Uses if-else statements to check if the number is greater than 0 (positive), less than 0 (negative), or equal to 0 (zero).

- These methods help determine if a number is positive, negative, or zero in JavaScript. Choose the method that best fits your coding style or specific use case.

Ques 23) Find the Character Is Vowel or Not?
Ans 23) In JavaScript, you can check whether a number is positive or negative by using simple conditional statements. Here is a few methods to achieve this:

Method 1: Using if-else Statements

```
<script>

function checkNumber(num) {
if (num > 0) {
    console.log(num + " is positive.");
} else if (num < 0) {
    console.log(num + " is negative.");
} else {
    console.log(num + " is zero.");
}


</script>
```

Ques 24) Write to check whether a number is negative, positive or zero?

Ans 24) to check whether a number is negative, positive, or zero in JavaScript, you can use a simple function with conditional statements. Here's an example function that performs this check:

```
<script>

function checkNumber(num) {
if (num > 0) {
    return num + " is positive.";
} else if (num < 0) {
    return num + " is negative.";
} else {
    return num + " is zero.";
}
}

// Example usage:
console.log(checkNumber(10)); // Output: 10 is positive.
console.log(checkNumber(-5)); // Output: -5 is negative.
console.log(checkNumber(0)); // Output: 0 is zero.

</script>
```

Explanation:

Function Definition: The check Number function takes a single argument num, which is the number to be checked.

Conditionals:

- if (num > 0): Checks if the number is greater than zero, indicating it is positive.
- else if (num < 0): Checks if the number is less than zero, indicating it is negative.
- else: If neither condition above is met, the number must be zero.
- Return Statements: Each condition returns a string indicating whether the number is positive, negative, or zero.

Running the Function:

1. checkNumber(10) will return "10 is positive."
2. checkNumber(-5) will return "-5 is negative."
3. checkNumber(0) will return "0 is zero."

This function is straightforward and covers all possible cases for determining if a number is positive, negative, or zero.

Ques 25) write to find number is even or odd using ternary operator in JS?

Ans 25) you can use the ternary operator in JavaScript to check if a number is even or odd. The ternary operator provides a concise way to perform conditional checks. Here's how you can do it:

```
<script>

function isEvenOrOdd(num) {
return (num % 2 === 0) ? num + " is even." : num + " is odd.";
}

// Example usage:
console.log(isEvenOrOdd(4)); // Output: 4 is even.
console.log(isEvenOrOdd(7)); // Output: 7 is odd.

</script>
```

Explanation:

Function Definition: The is Even or Odd function takes a single argument num, which is the number to be checked.

Ternary Operator:

- (num % 2 === 0) ? num + " is even." : num + " is odd.";
- This checks if the remainder of num divided by 2 is zero (num % 2 === 0). If true, it returns the string indicating that the number is even. If false, it returns the string indicating that the number is odd.
- Running the Function:
- Is Even or Odd (4) will return "4 is even."
- Is Even or Odd (7) will return "7 is odd."

The ternary operator provides a succinct and readable way to determine if a number is even or odd in JavaScript.

Ques 26) Write find maximum number among 3 numbers using ternary operator in JS?

Ans 26) You can find the maximum number among three numbers using the ternary operator in JavaScript by nesting the ternary operators. Here's how you can do it:

```
<script>

function findMax(num1, num2, num3) {
return (num1 > num2)
    ? (num1 > num3 ? num1 : num3)
    : (num2 > num3 ? num2 : num3);
}

// Example usage:
console.log(findMax(5, 10, 3)); // Output: 10
console.log(findMax(12, 7, 15)); // Output: 15
console.log(findMax(9, 9, 9)); // Output: 9

</script>
```

Explanation:

Function Definition: The findMax function takes three arguments num1, num2, and num3, which are the numbers to be compared.

Ternary Operator:

- (num1 > num2) ? (num1 > num3 ? num1 : num3) : (num2 > num3 ? num2 : num3);
- The outer ternary operator checks if num1 is greater than num2.
- If true, it then checks if num1 is also greater than num3. If num1 is greater, it returns num1; otherwise, it returns num3.
- If false, it then checks if num2 is greater than num3. If num2 is greater, it returns num2; otherwise, it returns num3.

Running the Function:

1. findMax(5, 10, 3) will return 10.
2. findMax(12, 7, 15) will return 15.
3. findMax(9, 9, 9) will return 9.

This method efficiently finds the maximum number among three given numbers using nested ternary operators in JavaScript.

Ques 27) Write to find minimum number among 3 numbers using ternary operator in JS?

Ans 27) To find the minimum number among three numbers using the ternary operator in JavaScript, you can use a similar approach to finding the maximum, but modify it to check for the smallest value. Here's how you can do it:

```
<script>

function findMin(num1, num2, num3) {
return (num1 < num2)
    ? (num1 < num3 ? num1 : num3)
    : (num2 < num3 ? num2 : num3);
}

// Example usage:
console.log(findMin(5, 10, 3)); // Output: 3
console.log(findMin(12, 7, 15)); // Output: 7
console.log(findMin(9, 9, 9)); // Output: 9

</script>
```

Explanation:

Function Definition: The find in function takes three arguments num1, num2, and num3, which are the numbers to be compared.

Ternary Operator:

- (num1 < num2) ? (num1 < num3 ? num1 : num3) : (num2 < num3 ? num2 : num3);
- The outer ternary operator checks if num1 is less than num2.
- If true, it then checks if num1 is also less than num3. If num1 is less, it returns num1; otherwise, it returns num3.
- If false, it then checks if num2 is less than num3. If num2 is less, it returns num2; otherwise, it returns num3.

Running the Function:

1. findMin(5, 10, 3) will return 3.
2. findMin(12, 7, 15) will return 7.
3. findMin(9, 9, 9) will return 9.

This method efficiently finds the minimum number among three given numbers using nested ternary operators in JavaScript.

Ques 28) Write to find the largest of three numbers in JS?

Ans 28) Certainly! To find the largest of three numbers in JavaScript, you can use a simple comparison using conditional statements (if-else). Here's an example of how to do it:

```
<script>

function findLargest(num1, num2, num3) {
let largest;

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

return largest;
}

// Example usage:
let num1 = 10;
let num2 = 20;
let num3 = 15;

console.log("The largest number is " + findLargest(num1, num2, num3));

</script>
```

- This function find Largest compares three numbers and returns the largest one. The example usage shows how to call this function and output the result. You can change the values of num1, num2, and num3 to test with different numbers.

Ques 29) Write to show:

I. Monday to Sunday using switch case in JS?

Ans 29) Certainly! You can use a switch statement in JavaScript to map numbers (1 to 7) to the corresponding days of the week (Monday to Sunday)

```
<script>
function getDayOfWeek(dayNumber) {
let dayName;
switch (dayNumber) {
    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 number";
}
return dayName;
}
let dayNumber = 3;
console.log("The day is " + getDayOfWeek(dayNumber));
</script>
```

- In this example, the function get Day of Week takes a number as input and uses a switch statement to return the corresponding day of the week. If the number is not between 1 and 7, it returns "Invalid day number". You can change the value of day Number to test with different inputs.

II. Vowel or Consonant using switch case in JS?

Ans) Certainly! You can use a switch statement to determine whether a given letter is a vowel or a consonant. Here's how you can do it:

```
    <script>
    function checkVowelOrConsonant(char) {
    let result;

    // Convert the character to lower case to handle both upper and lower case input
    char = char.toLowerCase();

    switch (char) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            result = "Vowel";
            break;
        default:
            result = "Consonant";
    }
    return result;
}

// Example usage:
let char = 'E';
console.log(char + " is a " + checkVowelOrConsonant(char));
        </script>
```

- In this example, the function check Vowel or Consonant takes a character as input, converts it to lowercase (to handle both uppercase and lowercase input), and uses a switch statement to check if the character is a vowel. If the character is a vowel, it returns "Vowel"; otherwise, it returns "Consonant". You can change the value of char to test with different letters.

(Conditional Looping Logic Questions)

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

Ans 30) JavaScript provides several looping structures to execute a block of code multiple times. The main looping structures are:

- for loop
- while loop
- do...while loop
- for...in loop
- for...of loop

Here's an example of each:

1. for Loop

The for loop is used when you know in advance how many times you want to execute a statement or a block of statements.

```
<script>
for (let i = 0; i < 5; i++) {
console.log("Iteration number " + i);
}
</script>
```

2. while Loop

The while loop is used when you want to execute a block of code as long as a specified condition is true.

```
<script>

let i = 0;
while (i < 5) {
console.log("Iteration number " + i);
i++;
}

</script>
```

3. do...while Loop

The do...while loop is similar to the while loop, but the block of code is executed at least once before the condition is tested.

```
<script>

let i = 0;
do {
console.log("Iteration number " + i);
i++;
} while (i < 5);

</script>
```

4. for...in Loop

The for...in loop is used to iterate over the enumerable properties of an object

```
<script>
const person = { fname: "John", lname: "Doe", age: 25 };

for (let key in person) {
console.log(key + ": " + person[key]);
}
</script>
```

5. for...of Loop

The for...of loop is used to iterate over iterable objects (like arrays, strings, etc.).

```
<script>
const array = ["a", "b", "c"];

for (let element of array) {
console.log(element);
}

</script>
```

Example of for Loop

Let's provide a detailed example using the for loop to iterate over an array:

```
<script>
const numbers = [1, 2, 3, 4, 5];

for (let i = 0; i < numbers.length; i++) {
console.log("Element at index " + i + " is " + numbers[i]);
}
</script>
```

- In this example, we use a for loop to iterate over the numbers array and print each element along with its index. The loop runs from i = 0 to i < numbers. length, which ensures that it processes each element in the array.

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

Ans 31) Sure! To print numbers from 972 to 897 using a for loop in JavaScript, you need to set up the loop to decrement the value in each iteration. Here's how you can do it:

```
<script>
for (let i = 972; i >= 897; i--) {
console.log(i);
}
</script>
```

- This loop starts at 972 and decrements i by 1 in each iteration until i is equal to 897. The console.log(i) statement prints the value of i in each iteration.

Ques 32) Write to print factorial of given number?

Ans 32) To calculate and print the factorial of a given number in JavaScript, you can use a for loop. Here's a function that takes a number as input and prints its factorial:

```
<script>
function factorial(num) {
if (num < 0) {
    console.log("Factorial is not defined for negative numbers");
    return;
}
let result = 1;
for (let i = 1; i <= num; i++) {
    result *= i;
}
console.log(`The factorial of ${num} is ${result}`);
}

// Example usage:
factorial(5); // Output: The factorial of 5 is 120\
</script>
```

In this code:

- The factorial function takes one argument num.
- It first checks if the number is negative and prints an appropriate message if so.
- It initializes result to 1.
- It uses a for loop to multiply result by each number from 1 to num.
- Finally, it prints the result.
- You can call this function with any positive integer to print its factorial.

Ques 33) Write to print Fibonacci series up to given numbers?

Ans 33) To print the Fibonacci series up to a given number in JavaScript, you can use a function that generates and prints the series. Here's an example of how you can do it:

```
<script>
function printFibonacciSeries(n) {
if (n <= 0) {
    console.log("Please enter a positive integer.");
    return;
}
let a = 0, b = 1, nextTerm;
console.log("Fibonacci Series:");
console.log(a); // print the first term
if (n > 1) {
    console.log(b); // print the second term
}
for (let i = 3; i <= n; i++) {
    nextTerm = a + b;
    console.log(nextTerm);
    a = b;
    b = nextTerm;
}
}
// Example usage:
printFibonacciSeries(10); // Output: Fibonacci series up to 10 terms
</script>
```

In this code:

- The printFibonacciSeries function takes one argument n, which is the number of terms in the Fibonacci series to print.
- It checks if the input is a positive integer and prints an error message if not.
- It initializes the first two terms of the Fibonacci series (a and b) to 0 and 1, respectively.
- It prints the first term (a). If n is greater than 1, it also prints the second term (b).
- It uses a for loop to calculate and print the next terms of the series up to the nth term.
- In each iteration, it calculates the next term as the sum of the previous two terms, updates the previous terms, and prints the next term.

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

Ans 34) To reverse a given number in JavaScript, you can convert the number to a string, split it into an array, reverse the array, and then join it back into a string. Finally, you can convert it back to a number if needed. Here's how you can do it:

```
<script>
function reverseNumber(num) {
// Convert the number to a string, split it into an array, reverse the array, and join it back
let reversedString = num.toString().split('').reverse().join('');
// Convert the reversed string back to a number
let reversedNumber = parseInt(reversedString);
return reversedNumber;
}
// Example usage:
let number = 64728;
let reversed = reverseNumber(number);
console.log(`Original number: ${number}`); // Output: Original number: 64728
console.log(`Reversed number: ${reversed}`); // Output: Reversed number: 82746
</script>
```

In this code:

- The reverseNumber function takes one argument num.
- It converts the number to a string using toString().
- It splits the string into an array of characters using split('').
- It reverses the array using reverse().
- It joins the array back into a string using join('').
- It converts the reversed string back to a number using parseInt().
- The function returns the reversed number.
- The example usage demonstrates how to call the function and print the original and reversed numbers.

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

Ans 35) To find the summation of the digits of a given number in JavaScript, you can convert the number to a string, split it into individual digits, convert those digits back to numbers, and then sum them up. Here's how you can do it:

```
<script>
function sumOfDigits(num) {
// Convert the number to a string, split it into an array of characters, then map each character back to a
number//
let digits = num.toString().split('').map(Number);
// Sum all the digits
let sum = digits.reduce((acc, digit) => acc + digit, 0);
return sum;
}
// Example usage:
let number = 1523;
let sum = sumOfDigits(number);
console.log(`The summation of digits of ${number} is ${sum}`); // Output: The summation of digits of 1523
is 11 //
</script>
```

In this code:

- The sumOfDigits function takes one argument num.
- It converts the number to a string using toString().
- It splits the string into an array of characters using split('').
- It maps each character back to a number using map(Number).
- It uses the reduce method to sum up all the digits in the array. The reduce method takes a callback function that adds each digit to an accumulator (acc), starting with an initial value of 0.
- The function returns the sum of the digits.

The example usage demonstrates how to call the function and print the original number and its digit summation.

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

Ans 36) To find the summation of the first and last digits of a given number in JavaScript, you can convert the number to a string, extract the first and last

characters, convert them back to numbers, and then sum them up. Here's how you can do it:

```
<script>
function sumFirstAndLastDigit(num) {
// Convert the number to a string
let numStr = num.toString();
// Get the first digit
let firstDigit = parseInt(numStr[0]);
// Get the last digit
let lastDigit = parseInt(numStr[numStr.length - 1]);
// Sum the first and last digits
let sum = firstDigit + lastDigit;
return sum;
}
// Example usage:
let number = 1234;
let sum = sumFirstAndLastDigit(number);
console.log(`The summation of the first and last digits of ${number} is ${sum}`); // Output: The summation
of the first and last digits of 1234 is 5//
</script>
```

In this code:

- The sumFirstAndLastDigit function takes one argument num.
- It converts the number to a string using toString().
- It extracts the first digit using numStr[0] and the last digit using numStr[numStr.length - 1].
- It converts these string digits back to numbers using parseInt().
- It sums the first and last digits.
- The function returns the sum.
- The example usage demonstrates how to call the function and print the summation of the first and last digits of the given number.

Ques 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

Ans 37) To print the specified pattern using console.log() and escape characters in JavaScript, you can use a loop to generate each line and print it. Here's how you can do it:

```
<script>
for (let i = 1; i <= 5; i++) {
console.log(`${i} 1 ${i} ${i**2} ${i**3}`);
}
</script>
```

In this code:

- We use a for loop to iterate from 1 to 5.
- Inside the loop, we use console.log() to print each line. We use template literals (enclosed in backticks `) to insert the values of i, i2(i squared), andi3` (i cubed) directly into the string.
- ${i} is a placeholder that gets replaced with the value of i in each iteration.

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

1) 1

1 0

1 0 1

1 0 1 0

1 0 1 0 1

Ans 38 (1)) To print the specified pattern using console.log() in JavaScript, you can use nested loops. Here's how you can do it:

```
<script>
for (let i = 1; i <= 5; i++) {
let line = '';
for (let j = 1; j <= i; j++) {
    line += (j % 2 === 1 ? '1 ' : '0 ');
}
console.log(line.trim());
}
</script>
```

2) A

B C

D E F

G H I J

K L M N O

Ans 38 (2)) To print the specified pattern using console.log() in JavaScript, you can use nested loops and keep track of the current letter. Here's how you can do it:

```
<script>
let currentCharCode = 65; // ASCII code for 'A'

for (let i = 1; i <= 5; i++) {
let line = '';
for (let j = 1; j <= i; j++) {
    line += String.fromCharCode(currentCharCode) + ' ';
    currentCharCode++;
}
console.log(line.trim());
}
</script>
```

3) 1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

Ans 38 (3)) To print the specified pattern using console.log() in JavaScript, you can use nested loops and keep track of the current number. Here's how you can do it:

```
<script>
let currentNumber = 1;
for (let i = 1; i <= 5; i++) {
let line = '';
for (let j = 1; j <= i; j++) {
    line += currentNumber + ' ';
    currentNumber++;
}
console.log(line.trim());
}
</script>
```

4) *

* *

* * *

* * * *

* * * * *

Ans 38 (4)) To print the specified pattern using console.log() in JavaScript, you can use nested loops to control the number of asterisks (*) printed in each row. Here's how you can do it:

```
<script>
for (let i = 1; i <= 5; i++) {
let line = '';
for (let j = 1; j <= i; j++) {
    line += '* ';
}
console.log(line);
}
</script>
```

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

Ans 39) Sure, here's a JavaScript program that accepts three numbers from the user, checks if each number is a palindrome, and then displays the result:

```
<script>
function isPalindrome(num) {
let originalNum = num;
let reversedNum = 0;
while (num > 0) {
    let digit = num % 10;
    reversedNum = reversedNum * 10 + digit;
    num = Math.floor(num / 10);
}
return originalNum === reversedNum;
}
function checkPalindrome() {
let count = 0;
while (count < 3) {
    let num = parseInt(prompt(`Enter number ${count + 1}:`));
    if (isNaN(num)) {
        alert('Invalid input! Please enter a valid number.');
        continue;
    }
    if (isPalindrome(num)) {
        console.log(`${num} is a palindrome.`);
    } else {
        console.log(`${num} is not a palindrome.`);
    }
    count++;
  }
 }
checkPalindrome();
</script>
```

In this program:

- The is Palindrome function checks whether a given number num is a palindrome or not. It reverses the number and compares it with the original number to determine if they are equal.
- The check Palindrome function repeatedly prompts the user to enter three numbers using a while loop. It checks each number using the is Palindrome function and displays the result.
- If the user enters a non-numeric value, the program alerts the user and prompts for input again.
- After checking all three numbers, the program ends.

Ques 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 ?

Ans 40) Certainly! You can use JavaScript's Date object to get the current day and time and then format it according to the given format. Here's how you can do it:

```
<script>
function getCurrentDay() {
const days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];
return days[new Date().getDay()];
}

function getCurrentTime() {
let date = new Date();
let hours = date.getHours();
let minutes = date.getMinutes();
let period = hours >= 12 ? 'PM' : 'AM';
hours = hours % 12;
hours = hours ? hours : 12; // Convert 0 to 12 for 12-hour clock
return `${hours} ${period}: ${date.getMinutes()} : ${date.getSeconds()} ${date.getMilliseconds()} `;
}

let currentDay = getCurrentDay();
let currentTime = getCurrentTime();

console.log(`Today is ${currentDay}. Current Time is ${currentTime}`);

</script>
```

In this code:

- The getCurrentDay function gets the current day of the week by using getDay() method of the Date object and then retrieves the corresponding day from an array of days.
- The getCurrentTime function gets the current time by using various methods (getHours(), getMinutes(), getSeconds(), and getMilliseconds()) of the Date object. It also adjusts the hours to be in 12-hour format and determines whether it's AM or PM.
- Finally, we call both functions to get the current day and time, and then print them in the desired format using console.log().

Ques 41) Write a JavaScript program to get the current date?

Ans 41) You can get the current date using JavaScript's Date object. Here's a simple JavaScript program to get the current date:

```
<script>
function getCurrentDate() {
let currentDate = new Date();
let day = currentDate.getDate();
let month = currentDate.getMonth() + 1; // Months are zero-based, so we add 1
let year = currentDate.getFullYear();

// Formatting single-digit day and month with leading zero if necessary
day = day < 10 ? '0' + day : day;
month = month < 10 ? '0' + month : month;

return `${year}-${month}-${day}`;
}

let currentDate = getCurrentDate();
console.log(`Current Date: ${currentDate}`);

</script>
```

Ques 42) Write a JavaScript program to compare two objects?

Ans 42) To compare two objects in JavaScript, you can write a function that iterates over the properties of both objects and compares their values. Here's one way to do it:

```
<script>
function compareObjects(obj1, obj2) {
// Get the keys of both objects
let keys1 = Object.keys(obj1);
let keys2 = Object.keys(obj2);
// Check if the number of keys is the same
if (keys1.length !== keys2.length) {
    return false;
}
// Iterate over the keys of the first object
for (let key of keys1) {
    // Check if the second object has the same key
    if (!obj2.hasOwnProperty(key)) {
        return false;
    }
    // Check if the values of the properties are equal
    if (obj1[key] !== obj2[key]) {
        return false;
    }
}
// If all checks pass, the objects are equal
return true;
}
// Example usage:
let obj1 = { name: 'John', age: 30 };
let obj2 = { name: 'John', age: 30 };
let obj3 = { name: 'Jane', age: 25 };
console.log(compareObjects(obj1, obj2)); // Output: true (obj1 and obj2 are equal)
console.log(compareObjects(obj1, obj3)); // Output: false (obj1 and obj3 are not equal)
</script>
```

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

Ans 43) You can convert an array of objects into a CSV (Comma-Separated Values) string by iterating over the array and extracting the values of each object's properties. Here's how you can do it:

```
<script>
function arrayToCSV(array) {
// Extract headers from the first object
let headers = Object.keys(array[0]);

// Create CSV string with headers
let csv = headers.join(',') + '\n';

// Iterate over each object in the array
array.forEach(obj => {
    // Extract values for each header
    let values = headers.map(header => obj[header]);
    // Join values with commas and add to CSV string
    csv += values.join(',') + '\n';
});

return csv;
}

// Example usage:
let data = [
{ name: 'John', age: 30, city: 'New York' },
{ name: 'Jane', age: 25, city: 'Los Angeles' },
{ name: 'Doe', age: 40, city: 'Chicago' }
];

let csvString = arrayToCSV(data);
console.log(csvString);
</script>
```

Ques 44) Write a JavaScript program to capitalize first letter of a string?

Ans 44) You can capitalize the first letter of a string in JavaScript by extracting the first letter, converting it to uppercase, and then concatenating it with the rest of the string. Here's a simple function to do that:

```
<script>
function capitalizeFirstLetter(str) {
// Check if the string is empty
if (str.length === 0) {
    return str;
}
// Capitalize the first letter and concatenate with the rest of the string
return str.charAt(0).toUpperCase() + str.slice(1);
}

// Example usage:
let inputString = 'hello world';
let capitalizedString = capitalizeFirstLetter(inputString);
console.log(capitalizedString); // Output: Hello world

</script>
```

In this code:

- The capitalize First Letter function takes a string str as input.
- It checks if the string is empty. If so, it returns the empty string.
- Otherwise, it extracts the first character of the string using char At(0), converts it to uppercase using to Upper Case(), and then concatenates it with the rest of the string using slice(1).
- The function returns the capitalized string.
- You can call this function with any string, and it will return the string with the first letter capitalized.

Ques 45) Write a JavaScript program to determine if a variable is array?

Ans 45) you can determine if a variable is an array in JavaScript using the Array .is Array () method. Here's how you can do it:

```
<script>
function isArray(variable) {
 return Array.isArray(variable);
 }

 // Example usage:
 let arr = [1, 2, 3];
 let notArr = 'not an array';

 console.log(isArray(arr));    // Output: true
 console.log(isArray(notArr)); // Output: false

</script>
```

In this code:

- The is Array function takes a variable variable as input.
- It uses the Array .is Array () method to check if the variable is an array.
- The function returns true if the variable is an array, and false otherwise.
- You can call this function with any variable, and it will return true if the variable is an array, and false otherwise.

Ques 46) Write a JavaScript program to clone an array?

Ans 46) You can clone an array in JavaScript using various methods, such as slice(), concat(), or the spread operator (...). Here's how you can do it using the spread operator:

```
<script>
function cloneArray(arr) {
 return [...arr];
 }

 // Example usage:
 let originalArray = [1, 2, 3];
 let clonedArray = cloneArray(originalArray);

 console.log(originalArray); // Output: [1, 2, 3]
 console.log(clonedArray);   // Output: [1, 2, 3]

 // Modify the original array
 originalArray.push(4);

 console.log(originalArray); // Output: [1, 2, 3, 4]
 console.log(clonedArray);   // Output: [1, 2, 3]

 </script>
```

In this code:

- The clone Array function takes an array is as input.
- It uses the spread operator (...) to create a new array with the same elements as the input array arr.
- The function returns the cloned array.
- You can call this function with any array, and it will return a new array with the same elements as the original array, effectively cloning it. Modifying the original array after cloning won't affect the cloned array.

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

Ans 47) Declaring methods directly in JavaScript objects, especially in scenarios where you're creating multiple instances of objects using constructor functions or classes, can have a drawback related to memory consumption and performance.

- When you define a method directly in an object, each instance of that object will have its own copy of that method. This means that if you have multiple instances of the object, each instance will consume memory to store its own copy of the method. This can lead to inefficient memory usage, especially if the method is large or if you have a large number of instances.
- In contrast, when you use prototype-based inheritance in JavaScript and define methods on the prototype of a constructor function or class, all instances share the same method. This results in more efficient memory usage because each instance doesn't need to store its own copy of the method.

Here's an example to illustrate the difference:

```
<script>

// Declaring method directly in object
let obj1 = {
method: function() {
    console.log("Method defined directly in object");
}
};

// Defining method on prototype
function MyClass() {}

MyClass.prototype.method = function() {
console.log("Method defined on prototype");
};

let obj2 = new MyClass();

</script>
```

In this example, obj1 has its method defined directly in the object, while obj2 has its method defined on the prototype of the MyClass constructor function. If you create multiple instances of obj1 or obj2, each instance of obj1 will have its own copy of the method, whereas all instances of obj2 will share the same

method defined on the prototype. This can lead to better memory usage and performance when dealing with a large number of instances.

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

Ans 48) To print the length of a string on the browser console using console.log(), you can simply pass the length property of the string to console.log(). Here's how you can do it:

```
<script>

let str = "Hello, World!";
console.log(str.length);

</script>
```

When you run this code in a browser environment and open the developer console, it will print the length of the string "Hello, World!" to the console.

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

Ans 49) You can change all the characters in a string to capital letters using the toUpperCase() method. Here's how you can do it:

```
<script>

let str = "hello, world!";
let capitalizedStr = str.toUpperCase();
console.log(capitalizedStr);

</script>
```

In this code:

- We first define a string str with the value "hello, world!".
- We then call the to Upper Case() method on the string str. This method returns a new string with all characters converted to uppercase.
- The result is stored in the variable capitalized Str.

- Finally, we log the capitalizedStr to the console, which will display "HELLO, WORLD!"

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

Ans 50) Methods directly in javsacript:

- Declaring methods directly in JavaScript objects can have a drawback related to memory consumption and performance, especially when you're creating multiple instances of objects using constructor functions or classes.
- When you define a method directly in an object, each instance of that object will have its own copy of that method. This means that if you have multiple instances of the object, each instance will consume memory to store its own copy of the method. This can lead to inefficient memory usage, especially if the method is large or if you have a large number of instances.
- In contrast, when you use prototype-based inheritance in JavaScript and define methods on the prototype of a constructor function or class, all instances share the same method. This results in more efficient memory usage because each instance doesn't need to store its own copy of the method.

Here's an example to illustrate the difference:

```
<script>

// Declaring method directly in object
let obj1 = {
method: function() {
    console.log("Method defined directly in object");
}
};

// Defining method on prototype
function MyClass() {}

MyClass.prototype.method = function() {
console.log("Method defined on prototype");
};

let obj2 = new MyClass();

</script>
```

- In this example, obj1 has its method defined directly in the object, while obj2 has its method defined on the prototype of the MyClass constructor function. If you create multiple instances of obj1 or obj2, each instance of obj1 will have its own copy of the method, whereas all instances of obj2 will share the same method defined on the prototype. This can lead to better memory usage and performance when dealing with a large number of instances.

Ques 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?

Ans 51) You can get the current date in the desired format (mm-dd-yyyy, mm/dd/yyyy, dd-mm-yyyy, or dd/mm/yyyy) using JavaScript's Date object and string manipulation. Here's how you can do it:

```
<script>
function getCurrentDate(format) {
 let currentDate = new Date();
 let day = currentDate.getDate();
 let month = currentDate.getMonth() + 1; // Months are zero-based, so we add 1
 let year = currentDate.getFullYear();
 day = day < 10 ? '0' + day : day;
 month = month < 10 ? '0' + month : month;
 switch (format) {
     case 'mm-dd-yyyy':
         return `${month}-${day}-${year}`;
     case 'mm/dd/yyyy':
         return `${month}/${day}/${year}`;
     case 'dd-mm-yyyy':
         return `${day}-${month}-${year}`;
     case 'dd/mm/yyyy':
         return `${day}/${month}/${year}`;
     default:
         return 'Invalid format';
 }
 }
 // Example usage:
 console.log(getCurrentDate('mm-dd-yyyy')); // Output: Current date in mm-dd-yyyy format
 console.log(getCurrentDate('mm/dd/yyyy')); // Output: Current date in mm/dd/yyyy format
 console.log(getCurrentDate('dd-mm-yyyy')); // Output: Current date in dd-mm-yyyy format
 console.log(getCurrentDate('dd/mm/yyyy')); // Output: Current date in dd/mm/yyyy format
 </script>
```

In this code:

- The getCurrentDate function takes a format parameter specifying the desired date format.
- It creates a new Date object to get the current date and time.
- It extracts the day, month, and year components from the date object.

- It pads single-digit day and month with a leading zero if necessary.
- It formats the date based on the provided format using a switch statement.
- The function returns the formatted date.

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

Ans 52) To determine the position of the first occurrence of a character (such as 'a') in a string in JavaScript, you can use the indexOf() method. Here's how you can do it for the string "30 Days Of JavaScript":

```
<script>
let str = "30 Days Of JavaScript";
let position = str.indexOf('a');
console.log("The position of the first occurrence of 'a' in the string is:", position);
</script>
```

In this code:

- We define a string str with the value "30 Days Of JavaScript".
- We use the indexOf() method on the string str to find the position of the first occurrence of the character 'a'.
- The result is stored in the variable position.
- Finally, we log the position to the console.

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

Ans 53) To determine the position of the last occurrence of a character (such as 'a') in a string in JavaScript, you can use the lastIndexOf() method. Here's how you can do it for the string "30 Days Of JavaScript":

```
<script>
let str = "30 Days Of JavaScript";
let position = str.lastIndexOf('a');

console.log("The position of the last occurrence of 'a' in the string is:", position);

</script>
```

In this code:

- We define a string str with the value "30 Days Of JavaScript".
- We use the lastIndexOf() method on the string str to find the position of the last occurrence of the character 'a'.
- The result is stored in the variable position.
- Finally, we log the position to the console.

Ques 54) Form Validtion in JS?

Ans 54)

- Form validation in JavaScript is a process of verifying that user input in a web form meets certain criteria before it is submitted to the server.
- This helps ensure that only valid data is sent to the server, reducing the chances of errors and improving the overall user experience.
- Here's a basic example of how you can perform form validation using JavaScript:

```
  <script>
 function validateForm() {
  let name = document.getElementById('name').value.trim();
  let email = document.getElementById('email').value.trim();
  // Reset error messages
  document.getElementById('nameError').textContent = '';
  document.getElementById('emailError').textContent = '';
  // Validate name
  if (name === '') {
      document.getElementById('nameError').textContent = 'Name is required';
      return false;
  }
  // Validate email
  if (email === '') {
      document.getElementById('emailError').textContent = 'Email is required';
      return false;
  } else if (!isValidEmail(email)) {
      document.getElementById('emailError').textContent = 'Invalid email format';
      return false;
  }
  // Form is valid
  return true;

unction isValidEmail(email) {
  // Regular expression for validating email format
  let emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);

  </script>
```

Ques 55) Form in Email, number, Password, Validation?

Ans 55) Sure, let's create a simple HTML form with fields for email, number, and password, and perform validation using JavaScript. Here's an example:

```
<script>
function validateForm() {
 // Get form inputs
 let email = document.getElementById('email').value.trim();
 let number = document.getElementById('number').value.trim();
 let password = document.getElementById('password').value.trim();

 // Reset error messages
 document.getElementById('emailError').textContent = '';
 document.getElementById('numberError').textContent = '';
 document.getElementById('passwordError').textContent = '';
 // Validate email
 if (email === '') {
     document.getElementById('emailError').textContent = 'Email is required';
     return false;
 } else if (!isValidEmail(email)) {
     document.getElementById('emailError').textContent = 'Invalid email format';
     return false;
 }
 // Validate number
 if (number === '') {
     document.getElementById('numberError').textContent = 'Number is required';
     return false;
 } else if (isNaN(number)) {
     document.getElementById('numberError').textContent = 'Invalid number';
     return false;
 }
// Validate password
if (password === '') {
    document.getElementById('passwordError').textContent = 'Password is required';
    return false;
} else if (password.length < 6) {
    document.getElementById('passwordError').textContent = 'Password must be at least 6 characters';
    return false;
}
// Form is valid
return true;
}

function isValidEmail(email) {
// Regular expression for validating email format
let emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
return emailRegex.test(email);
}

</script>
```

In this example:

- We have a simple HTML form with input fields for email, number, and password, and a submit button.
- When the form is submitted (onsubmit="return validateForm()"), the validateForm function is called.
- Inside validateForm, we get the values of the email, number, and password fields, trim whitespace, and reset any previous error messages.
- We then validate each field. If any validation fails, we display an error message next to the corresponding field and return false to prevent form submission.
- If all validations pass, we return true to allow form submission.
- The is Valid Email function uses a regular expression to validate the email format.
- You can customize the validation rules and error messages as needed.

Ques 56) Dynamic Form Validation in JS?

Ans 56) Dynamic form validation in JavaScript involves validating user input on the client-side before it is submitted to the server.

```
    return isUsernameValid && isEmailValid && isPasswordValid;
}
function validateUsername() {
    const usernameInput = document.getElementById("username");
    const usernameError = document.getElementById("usernameError");
    if (usernameInput.value.trim() === "") {
        usernameError.textContent = "Username is required.";
        usernameInput.classList.add("invalid");
        usernameInput.classList.remove("valid");
        return false;
    } else {
        usernameError.textContent = "";
        usernameInput.classList.add("valid");
        usernameInput.classList.remove("invalid");
        return true;
    }
}
function validateEmail() {
    const emailInput = document.getElementById("email");
    const emailError = document.getElementById("emailError");

    const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailPattern.test(emailInput.value)) {
        emailError.textContent = "Please enter a valid email address.";
        emailInput.classList.add("invalid");
        emailInput.classList.remove("valid");
        return false;
} else {
    emailError.textContent = "";
    emailInput.classList.add("valid");
    emailInput.classList.remove("invalid");
    return true;
}
}

function validatePassword() {
const passwordInput = document.getElementById("password");
const passwordError = document.getElementById("passwordError");

if (passwordInput.value.length < 6) {
    passwordError.textContent = "Password must be at least 6 characters long.";
    passwordInput.classList.add("invalid");
    passwordInput.classList.remove("valid");
    return false;
} else {
    passwordError.textContent = "";
    passwordInput.classList.add("valid");
    passwordInput.classList.remove("invalid");
    return true;
}
}
</script>
```

Explanation

- HTML Form: A form with username, email, and password fields. Each field has a corresponding for displaying error messages.
- Event Listeners: Event listeners are added to the input fields to call validation functions on input events.

Validation Functions:

- ❖ validateUsername(): Checks if the username field is not empty.
- ❖ validateEmail(): Validates the email format using a regular expression.
- ❖ validatePassword(): Ensures the password is at least 6 characters long.

- Real-time Feedback: The validation functions update the UI with error messages and add appropriate classes to input fields based on their validity.
- Form Submission: The form submission is prevented if any validation checks fail. If all checks pass, a success message is displayed (in a real application, the form would be submitted to the server).

Ques 57) how many type of JS Event? How to use it ?

Ans 57) Types of Javascript Events such as :

- ➢ Mouse events
- ➢ Keyboard events
- ➢ Form events
- ➢ Window events
- ➢ Clipboard events
- ➢ Drag & drop events
- ➢ Media events
- ➢ Touch events
- ➢ Focus events
- ➢ Miscellaneous events

JavaScript events can be handled using event listeners or event handler attributes in HTML.

Using Event Listeners

Event listeners are added to elements using the add Event Listener method. This method allows you to specify the type of event to listen for and the function to execute when that event occurs.

```
<button id="myButton">Click me</button>
<input type="text" id="myInput" placeholder="Type something...">

<script>
document.addEventListener("DOMContentLoaded", () => {
 const button = document.getElementById("myButton");
 const input = document.getElementById("myInput");

 // Mouse event
 button.addEventListener("click", () => {
     alert("Button clicked!");
 });

 // Keyboard event
 input.addEventListener("keydown", (event) => {
     console.log(`Key pressed: ${event.key}`);
 });

 // Form event
 input.addEventListener("input", () => {
     console.log(`Input value: ${input.value}`);
 });
});

</script>
```

Conclusion

- JavaScript events are a powerful way to make web pages interactive. By using event listeners, you can create a responsive and dynamic user experience. The add Event Listener method is particularly useful because it allows you to separate your JavaScript from your HTML, making your code more modular and easier to maintain.

Ques 58) What is Bom vs Dom in JS?

Ans 58) In JavaScript, BOM (Browser Object Model) and DOM (Document Object Model) are two crucial concepts that help in manipulating and interacting with web pages. Understanding the difference between BOM and DOM is fundamental for web development.

BOM (Browser Object Model)

- The Browser Object Model represents the browser window and provides objects and methods to interact with the browser. BOM allows you to manipulate the browser environment outside the content of the webpage, including things like the browser history, navigator, screen, and location.

Key Components of BOM

- ❖ Window Object: The global object that represents the browser window. All global JavaScript objects, functions, and variables are members of the window object.
- ❖ Example: window.alert("Hello!");

- ❖ Navigator Object: Contains information about the browser.
- ❖ Example: console.log(navigator.userAgent);

- ❖ Location Object: Contains information about the current URL and methods to redirect the browser to a new URL.

- ❖ Example: window.location.href = "https://www.example.com";

- ❖ History Object: Provides methods to interact with the browser's history.

- ❖ Example: window.history.back();

- ❖ Screen Object: Contains information about the user's screen.

- ❖ Example: console.log(screen.width);

```
<script>
// BOM Example
// Alert box
window.alert("Hello, World!");

// Redirecting to a new URL
window.location.href = "https://www.example.com";

// Logging browser information
console.log("Browser info: " + navigator.userAgent);

// Moving back one page in the history
window.history.back();

</script>
```

DOM (Document Object Model)

- The Document Object Model is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as a tree of nodes, where each node represents part of the document.

Key Components of DOM

- ❖ Document Object: Represents the entire HTML or XML document. It's the root of the document tree and provides methods to access and manipulate the document's content.
- ❖ Example: document.getElementById("myElement");

- ❖ Element Nodes: Represents HTML or XML elements. These nodes can be manipulated using various methods provided by the document object.

- ❖ Example: document.querySelector("p");

- ❖ Attributes Nodes: Represents attributes of HTML or XML elements. These can be accessed and manipulated through element nodes.

- ❖ Example:document.getElementById("myElement").getAttribute("class");

- Text Nodes: Represents the text content inside elements.

- Example: document.getElementById("myElement").textContent;

```
<script>
// DOM Example
// Accessing and modifying an element
let element = document.getElementById("myElement");
element.textContent = "Hello, DOM!";

// Creating a new element and appending it to the document
let newElement = document.createElement("div");
newElement.textContent = "New Div Element";
document.body.appendChild(newElement);

// Changing the style of an element
element.style.color = "red";

// Accessing all paragraph elements
let paragraphs = document.querySelectorAll("p");
paragraphs.forEach(p => p.style.backgroundColor = "yellow");

</script>
```

Ques 59) Array vs object defences in JS?

Ans 59) The difference between array Vs object is:

Objective	Array	Object
Defination	Arrays are ordered collections of elements, which can be of any type. Each element in an array has an index, starting from 0.	Objects are collections of key-value pairs where the keys (also known as properties) are strings (or symbols), and the values can be of any type.
Structure	Ordered list of elements accessed by numerical index.	Unordered collection of key-value pairs accessed by property names.
Use Case	Suitable for storing lists of items, especially when order matters.	Suitable for storing data with named properties, representing entities with attributes.
Methods	Comes with a rich set of methods for iterating, transforming, and	Methods are more focused on property management and general

	manipulating the list of elements.	object handling.
Performance	Generally more performant for operations that involve ordered lists, such as iteration or access by index.	May be more efficient for lookups by key, especially when using objects as maps or dictionaries.

Ques 61) Split the string into an array using split() Method?

Ans 61) In JavaScript, the split() method is used to divide a string into an array of substrings based on a specified separator. This method is very useful for breaking down a string into manageable parts. Here's a detailed explanation and examples of how to use the split() method:

Examples:

Example 1: Basic Usage

```
<script>
let str = "apple,banana,cherry";
let fruits = str.split(",");
console.log(fruits); // ["apple", "banana", "cherry"]
</script>
```

In this example, the string is split into an array of substrings using the comma (,) as the separator.

Example 2: Splitting with Space

```
<script>
let sentence = "Hello world, welcome to JavaScript!";
let words = sentence.split(" ");
console.log(words); // ["Hello", "world,", "welcome", "to", "JavaScript!"]
</script>
```

This example splits the string into an array of words using a space ( ) as the separator.

Example 3: Splitting into Characters

```
<script>
let str = "Hello";
let chars = str.split("");
console.log(chars); // ["H", "e", "l", "l", "o"]
</script>
```

Here, the string is split into an array of individual characters using an empty string ('') as the separator.

Example 4: Using a Limit

```
<script>
let str = "apple,banana,cherry,date";
let fruits = str.split(",", 2);
console.log(fruits); // ["apple", "banana"]
</script>
```

In this example, the string is split into an array with a maximum of 2 elements.

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

Ans 61) By using the includes() method, you can easily determine if a word or phrase exists within a given string:

```
<script>
let str = "JavaScript is a versatile language.";
let word = "Script";
if (str.includes(word)) {
console.log(`The string contains the word "${word}".`);
} else {
console.log(`The string does not contain the word "${word}".`);
}
// Output: The string contains the word "Script".
</script>
```

Ques 62) Change all the string characters to lowercase letters using to Lower Case () Method.

Ans 62) You can use toLowerCase() to convert both strings to lowercase before comparing them:

```
<script>
let str1 = "Hello World";
let str2 = "hello world";

if (str1.toLowerCase() === str2.toLowerCase()) {
console.log("The strings are equal (case-insensitive comparison).");
} else {
console.log("The strings are not equal.");
}
// Output: The strings are equal (case-insensitive comparison).

</script>
```

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

Ans 63) To find the character at a specific index in a string in JavaScript, you can use the charAt() method. Here's how you can find the character at index 15 in the string "30 Days of JavaScript":

```
<script>
string.charAt(index)
let str = "30 Days of JavaScript";
let characterAt15 = str.charAt(15);
console.log(`The character at index 15 is: '${characterAt15}'`); // The character at index 15 is: 'S'
</script>
```

Conclusion

The charAt() method is a simple and effective way to retrieve a character at a specific position in a string.

Ques 64) copy to one string to another string in JS?

Ans 64) Copying strings in JavaScript are a simple and efficient process due to the immutability of strings. Here’s a quick summary with a code example:

```
<script>
// Basic string copy
let originalString = "Hello, World!";
let copiedString = originalString;

console.log(copiedString); // "Hello, World!"

// Copying strings with a function
function copyString(str) {
return str;
}

let anotherString = "30 Days of JavaScript";
let anotherCopiedString = copyString(anotherString);

console.log(anotherCopiedString); // "30 Days of JavaScript"

// Copying strings in objects
let originalObject = { text: "30 Days of JavaScript" };
let copiedObject = { text: originalObject.text };

console.log(copiedObject.text); // "30 Days of JavaScript"
</script>
```

Ques 65) Find the length of a string without using library Function?

Ans 65) To find the length of a string without using any built-in library functions like length, you can use a loop to iterate through the string and count the number of characters. Here's how you can do it:

```
<script>
function getStringLength(str) {
let length = 0;
while (str[length] !== undefined) {
    length++;
}
return length;
}

let str = "30 Days of JavaScript";
let lengthOfString = getStringLength(str);

console.log(`The length of the string is: ${lengthOfString}`); // The length of the string is: 21
</script>
```

Explanation

- Function Definition: Define a function getStringLength that takes a string str as its parameter.
- Initialize Length: Initialize a variable length to 0. This will be used to count the characters in the string.
- Loop Through String: Use a while loop to iterate through the string. The loop continues as long as str[length] is not undefined.
- The condition str[length] !== undefined checks if there is a character at the current index.
- Increment Length: Inside the loop, increment the length variable by 1.
- Return Length: Once the loop ends, return the length variable which now holds the count of characters in the string.

(Basic questions Marked with star)

Ques 1) What is JavaScript?

Ans 1) A JavaScript is a high-level, interpreted programming language primarily used for creating and controlling dynamic website content.

Ques 2) What is the use of isNaN function?

Ans 2) A The isNaN function in JavaScript is used to determine whether a value is NaN (Not-a-Number). It checks if the value is not a valid number, and returns true if the value is NaN, and false.

Ques 3) What is negative Infinity?

Ans 3) A Negative Infinity in JavaScript is a special value that represents a value less than any other number. It is a property of the global Number object and is returned when a negative number is divided by zero, or when a mathematical operation results in an overflow to a negative value that exceeds the smallest possible negative number representable in JavaScript.

Ques 4) Which company developed JavaScript?

Ans 4) A JavaScript was developed by Netscape Communications Corporation. The language was created by Brendan Eich in 1995 during his time at Netscape. Initially, it was called Mocha, then renamed to LiveScript, and finally to JavaScript.

Ques5) What are undeclared and undefined variables?

Ans 5) A In JavaScript, undeclared and undefined variables refer to different situations related to variable declarations and their values.

Undeclared Variables	Undefined Variables
Undeclared variables are those that have not been declared using var, let, or const. When you try to use an undeclared variable, JavaScript throws a Reference Error.	Undefined variables are those that have been declared but have not been assigned a value. When you declare a variable without initializing it, its default value is undefined.

Ques6) Write the code for adding new elements dynamically?

Ans6)

```html
<!DOCTYPE html>
<html>
<head>
    <title>Dynamic Element Addition</title>
</head>
<body>
    <h1>Dynamic List</h1>
    <ul id="myList">
        <li>Item 1</li>
        <li>Item 2</li>
    </ul>
    <button id="addButton">Add Item</button>

    <script>
        // Function to add a new item to the list
        function addNewItem() {
            // Create a new list item element
            const newItem = document.createElement('li');
            // Create a text node with the item content
            const textNode = document.createTextNode('New Item');
            // Append the text node to the list item
            newItem.appendChild(textNode);
            // Find the unordered list element by its ID
            const list = document.getElementById('myList');
            // Append the new list item to the unordered list
            list.appendChild(newItem);
        }
        // Add an event listener to the button to call addNewItem on click
        document.getElementById('addButton').addEventListener('click', addNewItem);
    </script>
</body>
</html>
```

Ques 7) What is the difference between View State and Session State?

Ans 7) A View State and Session State are both used to manage state in web applications, but they differ in scope, storage, and use cases.

Differnce	View state	Session state
Scope	Limited to a single page.	Spans multiple pages within a user session.
storage	Stored on the client side in a hidden field within the page.	Stored on the server side.
Cases	Maintains the state of controls between postbacks on the same page.	Stores user-specific data like login info and preferences that need to persist across multiple pages.

Ques 8) What is === operator?

Ans 8) A The === operator in JavaScript is the strict equality operator. It checks for both value and type equality without performing type conversion. If the values and types are exactly the same, it returns true; otherwise, it returns false.

Examples:

1 === 1 // true

1 === '1' // false

true === 1 // false

null === null // true.

Ques9) How can the style/class of an element be changed?

Ans 9) A You can change the style or class of an element using JavaScript or CSS. In JavaScript, you can use methods like element .style .property = value to change inline styles or element. Class List. add/remove("class") to modify classes. In CSS, you can target elements by their IDs, classes, or tags and change their styles using properties like color, background- color, etc.

Ques 10) How to read and write a file using JavaScript?

Ans 10) In JavaScript, you can read and write files using the File Reader and File Writer APIs. For reading files, you can use the File Reader API, which

provides methods like read As Text () for reading text files, read As DataURL() for reading data as a URL, and read As Array Buffer() for reading binary files. For writing files, you can use the File Writer API, but it's only available in some environments like Node.js or in browser-based applications with certain restrictions due to security concerns. Alternatively, in Node.js, you can use the built-in fs module to read and write files synchronously or asynchronously.

Ques 11) What are all the looping structures in JavaScript?

Ans 11) All the looping structures in js are:

- for loop: Executes a block of code a specified number of times.
- while loop: Executes a block of code as long as the condition is true.
- do…while loop: Similar to a while loop, but the code block is executed once before checking the condition.
- for…of loop: Iterates over iterable objects like arrays, strings, etc.

Ques 12) How can you convert the string of any base to an integer in JavaScript?

Ans 12) A To convert a string of any base to an integer in JavaScript, you can use the parseInt() function along with the base as the second argument. For example, to convert a binary string to an integer, you would do:

```
<script>
let binaryString = "1010";
let decimalNumber = parseInt(binaryString, 2);
console.log(decimalNumber); // Output: 10
</script>
```

Similarly, you can convert strings representing numbers in octal, hexadecimal, or any other base by specifying the appropriate base in the parseInt() function.

Ques 13) * What is the function of the delete operator?

Ans 13) The delete operator in JavaScript is used to delete a property from an object. It can also be used to delete an element from an array, but it leaves a hole in the array (the deleted element will be undefined).

Example of deleting a property from an object:

```
<script>
let obj = { a: 1, b: 2, c: 3 };
delete obj.b;
console.log(obj); // Output: { a: 1, c: 3 }

let arr = [1, 2, 3, 4, 5];
delete arr[2];
console.log(arr); // Output: [1, 2, undefined, 4, 5]
</script>
```

It's important to note that delete only removes the property or element, but does not affect the length of an array. If you want to remove an element from an array without leaving a hole, you should use the splice() method.

Ques 14) What are all the types of Pop up boxes available in JavaScript?

Ans 14) A JavaScript provides three types of popup boxes:

- Alert: Displays a message to the user with an OK button.
- Confirm: Displays a message to the user with OK and Cancel buttons.
- Prompt: Prompts the user to enter input with a text field and OK and Cancel buttons.

Ques 15) What is the use of Void (0)?

Ans 15) A the void(0) expression is typically used to generate a undefined value in JavaScript. It's commonly seen in anchor tags (<a>) to prevent the browser from navigating to a new page when the anchor is clicked. For example:

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

Clicking on this link won't cause the browser to navigate anywhere because void(0) returns undefined, effectively cancelling the default behavior of the anchor tag.

Ques 16) How can a page be forced to load another page in JavaScript?

Ans 16) In JavaScript, you can force a page to load another page by changing the window.location property to the URL of the page you want to navigate to. There are a few different ways to accomplish this:

- Using window.location.replace ()

```
<script>
    window.location.replace("https://www.example.com");
</script>
```

The replace() method of the window.location object navigates to the specified URL by replacing the current page in the browser's history. This means that the user cannot navigate back to the original page using the browser's back button.

- Using window.location.assign()

```
<script>
    window.location.assign("https://www.example.com");
</script>
```

The assign() method of the window.location object also navigates to the specified URL, but it adds a new entry to the browser's history. This allows the user to navigate back to the original page using the browser's back button.

Ques 17) What are the disadvantages of using innerHTML in JavaScript?

Ans 17) Using innerHTML in JavaScript has several disadvantages:

I. **Security Risks:** Susceptible to Cross-Site Scripting (XSS) attacks if user input is not properly sanitized.
II. **Performance Issues:** Causes re-rendering of the entire element content and loss of event listeners.

III. **Maintainability:** Leads to harder-to-read code and mixes HTML with JavaScript, complicating debugging and maintenance.
IV. **SEO and Accessibility:** Dynamic content can be problematic for search engines and assistive technologies.
V. **Inconsistent Behavior:** Potential cross-browser compatibility issues, especially with older browsers.