WD - JavaScript Essentials &Advanced

Q.1 What is JavaScript. How to use it?
Ans: JavaScript is a scripting or programming language for dynamically updating content,control multimedia,animate images.
it is generally used for create user-friendly and interactive websites.

Q.2 How many type of Variable in JavaScript?
Ans:there are generally three types of variable in JavaScript:
1) let
2) Var
3) Const

Q.3 Define a Data Types in js?
Ans:
There are mainly two types in js:
1) Primitive Data Types:
    Number
    Null
    Boolean
    Big Int
    String
    Symbol
    Undefined

2) Non-Primitive Data Types:
    Object
    Array
    Function
    Date
    Regex

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

Ans:

```
function mul(a,b,c){
var multi=a*b*c;
return multi
 }
document.write(mul(10,20,30))
```

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

Ans:

Undefined:

A variable that has been declared but has not been assigned a value.

Exists within the scope it was declared

Attempting to access an undefined variable will not cause an error,it simply returns ‘undefined’.

Undeclare:

A variable that has not been declared in any scope.

Does not exist in any scope until it is first assigned a value.
If it is assigned without being declared,it implicitly becomes a global variable.
Attempting to access an undefined variable will result in a ‘ReferenceError’.

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

Ans:

```
<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("Mother Teresa once said, \"If you can't feed a hundred people, then feed just one.\"");
</script>
```

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

Ans:

```
<script>
```

```
   console.log(typeof('10'));
   console.log(typeof(10));
   let str = '10';
   num=10;
   console.log(str === num);
   console.log(Number(str)===num);
</script>
```

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

Ans:function area(b,h){

```
      a=0.5*b*h;
      Return a;
      }
      Console.log(area(10,10));
```

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

Ans:

```
      // Function to calculate the number of days
      left until next Christmas
      function daysUntilChristmas() {
      // Get the current date
      let currentDate = new Date();

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

```
// Create a new Date object for Christmas of
the current year
let christmasDate = new Date(currentYear, 11,
25); // Month is 0-based index (11 represents
December)

 // If Christmas has already passed this year,
set it to next year
 if (currentDate.getMonth() === 11 &&
currentDate.getDate() > 25) {
   christmasDate.setFullYear(currentYear + 1);
 }

// Calculate the difference in milliseconds
between the current date and Christmas
let timeDiff = christmasDate.getTime() -
currentDate.getTime();

// Calculate the number of days left until
Christmas
let daysLeft = Math.ceil(timeDiff / (1000 *
3600 * 24));

return daysLeft;

}
let daysLeft = daysUntilChristmas();
```

```
	console.log("Days left until next
	Christmas:", daysLeft);
```

Q.10 What is Condition Statement?

Ans:A condition statement, often referred to in programming contexts, is a fundamental concept used to control the flow of execution in a program based on whether certain conditions are true or false. It typically involves the use of conditional expressions or statements that evaluate to true or false (Boolean values).

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

Ans: Condition Statement is provide true or false a specified condition.

```
Function Rec(a){
	C=4*a;
	Return C;
}
Console.log(Rec(10));
```

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

Ans:

```
<script>
 function convert(year, day) {
    // Convert years to days
    var yearsToDays = year * 365.25;
    // Convert days to years
    var daysToYears = day / 365.25;
    return {
        days: yearsToDays,
        years: daysToYears
    };
}
result=convert(2, 214);
document.write(`2 years is approximately
${result.days.toFixed(2)} days.<br>`);
document.write(`214 days is approximately
${result.years.toFixed(2)} years.`);
</script>
```

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

Ans:

```
Function F2C(f):
      {
            C=(f-32)*5/9;
            Return C;
      }
      Console.log(F2C(40));
```

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

Ans:

```
<script>
        let file= prompt("Enter a file:");
        document.write(file.split('.').pop())
</script>
```

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

Ans: True

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

Ans:"hello"

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

Ans: False

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

Ans: Loop is repeatedly executes a block of code as long as a specified condition is true .

It is used to execute one block of code from multiple options based on the value of a given expression.

```
Switch(expression){
    case value1:
        //block of code
        Break;
    case value2:
        //block of code
        Break;
    Default;
        // Code to execute when none of the
        cases are matched
}
```

Q.19 What is the use of is Nan function?
Ans: It first tries to convert the given value to a number, and then checks if the result is NaN. This means it might return true for values that are not strictly NaN but are not valid numbers either.

Q.20 What is the difference between && and || in JavaScript?
Ans:
The && operator returns true if both operands are true; otherwise, it returns false.
The || operator returns true if at least one of the operands is true.

Q.21 What is the use of Void (0)?
Ans: The void keyword in JavaScript is used to evaluate an expression and then return undefined.

The most common use of void is in combination with the value 0 to create a self-executing anonymous function or to prevent a browser from navigating to a new page when clicking on a link.

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

Ans:

```
<script>
let num = prompt("Enter a number:");
num = Number(num);

function check(num){
return  (num > 0) ? "positive" : "negative";
}
```

Q.23 Find the Character Is Vowel or Not ?

Ans:

```
char="s"
function checkVowelOrConsonant(char) {
   switch(char) {
      case 'a':
      case 'e':
      case 'i':
      case 'o':
      case 'u':
         return char + " is a vowel";
```

```
            default:
                return char + " is a consonant";
        }
    }
document.write(checkVowelOrConsonant(char))
```

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

Ans:

```
<script>
        let num = prompt("Enter a number:");
        num = Number(num);

    function check(num){
        return num==0?num + " is zero.":
        (num>0?num+ "is positive.":num<0?num+ "is
        negative.")
        }
```

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

Ans:

```
<script>
    let inputNumber = prompt("Enter a number:");
    inputNumber = Number(inputNumber);

    function check(inputNumber){
        return(inputNumber%2==0) ? "Even":"Odd";
    }
    document.write(check(inputNumber))
</script>
```

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

Ans:

```
        <script>
        let a = 5, b = 10, c = 8;
        let max = (a > b) ? (a > c ? a : c) : (b > c ? b : c);
        console.log("The maximum number is "+max);
        </script>
```

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

Ans:

```
        <script>
        let a = 5, b = 10, c = 8;
        let max = (a <b) ? (a < c ? a : c) : (b < c ? b : c);
        console.log("The maximum number is "+max);
        </script>
```

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

Ans:

```
function findLargest(a, b, c) {
   if (a >= b && a >= c) {
      return a;
   } else if (b >= a && b >= c) {
      return b;
   } else {
      return c;
   }
}
const a = 10;
const b = 25;
const c = 15;

const largest = findLargest(a,b,c);
console.log("The largest number is " + largest);
```

Q.29 Write to show

i. Monday to Sunday using switch case in JS?

Ans:var date = new Date().getDay();

```
switch(date) {
   case 0:
      date = "Sunday";
      break;
   case 1:
      date = "Monday";
```

```
        break;
    case 2:
        date = "Tuesday";
        break;
    case 3:
        date = "Wednesday";
        break;
    case 4:
        date = "Thursday";
        break;
    case 5:
        date = "Friday";
        break;
    case 6:
        date = "Saturday";
        break;
    default:
        date = "Invalid day";
}


console.log("Today is " + date);
```

ii. Vowel or Consonant using switch case in JS?

Ans:

```
        <script>
        var char = "s";

        switch (char) {
            case 'a':
```

```
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            document.write(char + " is a vowel");
            break;
        default:
            document.write(char + " is a consonant");
    }
    </script>
```

Q.21 What are the looping structures in JavaScript? Any one Example?

Ans: There are many looping structures that allows to execute a block of code multiple times.

1) for loop
2) While loop
3) Do while loop
4) For in
5) For each
6) Switch

Example:

```
For(let i=0; I<100;I++){
        If(I%2==0){
                Docment.write(I + "<br>")
        }
}
```

Q.22 Write a print 972 to 897 using for loop in JS?

Ans:

```
	For(var I=972;I<897;I--){
		Document.write(I)
	}
```

Q.23 Write to print factorial of given number?

Ans: var I;

```
	 var  fact=0;
	var num=5;
	For(I,I<=n;I++){
		fact=fact*i
	}
	Console.log(fact)
```

Q.24 Write to print Fibonacci series up to given numbers?

Q.25 Write to print number in reverse order e.g.: number = 64728 ---> reverse =82746 in JS?

Ans:

```
function reverseNumber(number) {
var numberString=number.toString();
reversedString = ' ';
 for (var i = numberString.length - 1; i >= 0; I--) {
	reversedString=
	reversedString+numberString[i];
	}
 var reversedNumber = parseInt(reversedString);
return reversedNumber;
 }
var number = 64728;
console.log("Original number: " + number);
console.log("Reversed number: " +
reverseNumber(number));
```

Q.26 Write a program make a summation of given number (E.g., 1523 Ans: - 11) in JS?

```
function summation(num) {
   var sum = 0;
   var numString = num.toString();

   for (var i = 0; i < numString.length; i++) {
      sum += parseInt(numString[i]);
   }


   return sum;
}


var num = 1523;
var result = summation(num);
console.log(result);
```

Q.27 Write a program you have to make a summation of first and last Digit. (E.g., 1234 Ans: - 5) in JS?

```
Var num=1234;
Var a;
Var  sum=0;
Function sum(num){
       Var numstring=num.toString();
       For (var I=0;I<numstring.length;I++){


       sum=parseInt(numstring[0])+parseInt(numstri
       ng[length-1]);
}
return sum;
}
Console.log("sum");
```

Q.28 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
```

```
for(let i = 1; i <= 5; i++) {
       let output = "";
          for (let j = 0; j <= 3; j++) {
           output += `${power} `;
       }
       output = ${i} ${output};
```

```
        console.log(output.trim());
    }
```

Q.29 Use pattern in console.log in JS?

```
1
1 0
1 0 1
1 0 1 0
1 0 1 0 1
```

```
for (var i = 1; i <= 5; i++) {
    for (var j = 1; j <= i; j++) {
        if (j % 2 !== 0) {
            document.write("1 ");
        } else {
            document.write("0 ");
        }
    }
    document.write("<br>");
}
```

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

```
}

var currentCharCode = 65;
var pattern = [1, 3, 2, 4, 5];

for (var i = 0; i < pattern.length; i++) {
   for (var j = 0; j < pattern[i]; j++) {
      document.write(String.fromCharCode(current
CharCode) + " ");
      currentCharCode++;
   }
   document.write("<br>");
}
```

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

```
for (var i = 1; i <= 5; i++) {
   for (var j = 1; j <= i; j++) {
      document.write(sum + " ");
      sum++;
   }
   document.write("<br>");
}
```

```
*
* *
* * *
* * * *
* * * * *
```

```
for (var i = 1; i <= 5; i++) {
   for (var j = 1; j <= i; j++) {
      document.write("*" + " ");
   }
   document.write("<br>");
}
```

Q.30 Accept 3 numbers from user using while loop and check each numbers palindrome ?

To accept three numbers from the user using a while loop and check if each number is a palindrome in JavaScript, you can follow this approach:

1. Use a while loop to keep accepting input from the user.
2. For each number, check if it is a palindrome by reversing the number and comparing it with the original.

Here’s a JavaScript code example:

javascript

Copy code

```
function isPalindrome(num) {
// Convert number to string to easily reverse it
let str = num.toString();
let reversedStr = str.split('').reverse().join('');

// Compare original string with the reversed string
return str === reversedStr;
}

let count = 0;
let numbers = [];

// Accept 3 numbers from the user using a while loop
while (count < 3) {
let input = prompt("Enter a number:");
let num = parseInt(input);

// Add the number to the list
numbers.push(num);

// Check if the number is a palindrome
if (isPalindrome(num)) {
console.log(num + " is a palindrome.");
} else {
console.log(num + " is not a palindrome.");
}

// Increment the count
count++;
}

console.log("You entered the numbers: " +
numbers.join(", "));
```

Explanation:

1. **isPalindrome Function**: This checks if a number is a palindrome by converting the number to a string, reversing the string, and comparing it with the original string.
2. **while Loop**: The loop runs three times, each time prompting the user for a number.
3. **Palindrome Check**: Each input number is checked to see if it's a palindrome, and the result is logged to the console.

Make sure to run this code in a browser or an environment that supports prompt() for input.

Q.31 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 ?

To display the current day and time in the format shown, you can use JavaScript's Date object to retrieve the current day and time. You can then format the output accordingly.

Here’s a JavaScript program that outputs the current day and time in the desired format:

javascript
Copy code

```
function formatTime(date) {
// Get hours, minutes, and seconds from the Date
object
let hours = date.getHours();
let minutes = date.getMinutes();
let seconds = date.getSeconds();
let period = hours >= 12 ? 'PM' : 'AM';

// Convert to 12-hour format
hours = hours % 12;
hours = hours ? hours : 12; // Adjust for midnight (0
becomes 12)

// Pad minutes and seconds with leading zeros if
needed
minutes = minutes < 10 ? '0' + minutes : minutes;
seconds = seconds < 10 ? '0' + seconds : seconds;

return `${hours} ${period} : ${minutes} : ${seconds}`;
}

function displayCurrentDayAndTime() {
let date = new Date();

// Array of days to get the current day
let daysOfWeek = ["Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday"];
let today = daysOfWeek[date.getDay()];

// Get formatted time
let currentTime = formatTime(date);

// Display output
console.log(`Today is ${today}.`);
console.log(`Current Time is ${currentTime}`);
}

displayCurrentDayAndTime();
```

Explanation:

1. **formatTime() Function**: This formats the time in the desired 12-hour format with AM/PM. It also ensures minutes and seconds are padded with leading zeros if they are less than 10.
2. **displayCurrentDayAndTime() Function**: This retrieves the current date using the Date object, extracts the current day, formats the time using the formatTime() function, and logs the output.

Sample Output:

If you run the code on a Friday at 12:22:22 PM, the output would be:

```
csharp
Copy code
Today is Friday.
```

```
Current Time is 12 PM : 22 : 22
```

This program provides the desired format for the current day and time.

Q.32 Write a JavaScript program to get the current date?

To get the current date in JavaScript, you can use the built-in Date object and format it as needed. Here's a simple JavaScript program to display the current date:

javascript
Copy code

```
function getCurrentDate() {
let date = new Date();

// Get day, month, and year
let day = date.getDate();
let month = date.getMonth() + 1; // Months are zero-based, so add 1
let year = date.getFullYear();

// Format the date as DD/MM/YYYY
let formattedDate = `${day}/${month}/${year}`;

console.log("Current Date: " + formattedDate);
}
```

```
getCurrentDate();
```

Explanation:

1. **new Date()**: This creates a new Date object with the current date and time.
2. **getDate()**: Retrieves the day of the month.
3. **getMonth()**: Retrieves the month (zero-based, so January is 0 and December is 11).
4. **getFullYear()**: Retrieves the full year.
5. The date is formatted in DD/MM/YYYY format.

Sample Output:

If today is October 2, 2024, the output will be:

sql
Copy code
```
Current Date: 2/10/2024
```

Q.33 Write a JavaScript program to compare two objects?

To compare two objects in JavaScript, you need to ensure that both objects have the same properties with the same values. Since objects are reference types in JavaScript, comparing them directly with == or === will only compare their references, not their contents.

To compare the contents of two objects, you can create a function that checks the properties and values of both objects. Here’s a program that does that:

javascript
Copy code
```
function areObjectsEqual(obj1, obj2) {
// Check if both are objects
if (typeof obj1 !== 'object' || typeof obj2 !== 'object'
|| obj1 === null || obj2 === null) {
return false;
}

// Get the keys of both objects
let keys1 = Object.keys(obj1);
let keys2 = Object.keys(obj2);

// Compare the number of keys (properties)
if (keys1.length !== keys2.length) {
return false;
}

// Compare the values of corresponding keys
for (let key of keys1) {
// Check if the value is an object and recurse if
necessary
if (typeof obj1[key] === 'object' && typeof obj2[key]
=== 'object') {
if (!areObjectsEqual(obj1[key], obj2[key])) {
return false;
}
} else if (obj1[key] !== obj2[key]) {
return false;
}
}

return true;
}

// Example usage
let obj1 = {
name: "John",
age: 30,
details: {
country: "USA",
city: "New York"
}
};

let obj2 = {
name: "John",
age: 30,
details: {
country: "USA",
city: "New York"
}
};

console.log(areObjectsEqual(obj1, obj2)); // Output:
true

let obj3 = {
name: "John",
age: 30,
details: {
country: "USA",
city: "Los Angeles"
}
};

console.log(areObjectsEqual(obj1, obj3)); // Output: false
```

Explanation:

1. **areObjectsEqual() Function**:
    - First, it checks if both inputs are non-null objects.
    - Then, it compares the number of keys (properties) in both objects.
    - It iterates through the keys and compares the values of each key.
    - If the value is also an object, the function calls itself recursively to handle nested objects.
2. **Example Usage**:
    - Two objects (obj1 and obj2) are compared and found to be equal.
    - A third object (obj3) is compared to obj1 but has a different value for the city property, so they are considered unequal.

Output:

bash

Copy code

```
true
```

false

This approach compares both simple and nested objects.

4o

Q.34 Write a JavaScript program to convert an array of objects into CSV string?

To convert an array of objects into a CSV (Comma-Separated Values) string in JavaScript, you can follow these steps:

1. Extract the keys (header) from the first object in the array.
2. Convert each object into a row by mapping its values to a CSV-friendly format.
3. Join the headers and rows together to form the CSV string.

Here’s a JavaScript program to do this:

javascript
Copy code

```
function arrayToCSV(data) {
// Check if the input is an array and not empty
if (!Array.isArray(data) || data.length === 0) {
```

```
return '';
}

// Get the headers from the first object (keys of the
object)
const headers = Object.keys(data[0]);

// Map the data to CSV rows
const rows = data.map(obj => {
return headers.map(header => {
// Handle undefined or null values by replacing them
with an empty string
return obj[header] !== undefined && obj[header] !==
null ? obj[header] : '';
}).join(',');
});

// Combine headers and rows into a single CSV string
const csvString = [headers.join(','), ...rows].join('\n');

return csvString;
}

// Example usage
const data = [
{ name: "John", age: 30, city: "New York" },
{ name: "Jane", age: 25, city: "Los Angeles" },
{ name: "Sam", age: 35, city: "Chicago" }
];
```

```
const csvResult = arrayToCSV(data);
console.log(csvResult);
```

Explanation:

1. **Headers**:
   - We extract the keys from the first object in the array using Object.keys(). This will serve as the header row of the CSV.
2. **Mapping Objects to Rows**:
   - Each object in the array is mapped to a CSV row. We loop through the headers to ensure that all objects follow the same column order. If a value is undefined or null, it is replaced with an empty string.
3. **CSV String**:
   - The header and rows are joined together using newline characters (\n) to form the CSV string.

Example Output:

For the data array provided above, the output will be:

sql
Copy code

```
name,age,city
John,30,New York
Jane,25,Los Angeles
Sam,35,Chicago
```

This CSV string can be exported or used in other contexts, such as downloading a file or sending data to an API.

Q.35 Write a JavaScript program to capitalize first letter of a string?

To capitalize the first letter of a string in JavaScript, you can use the following approach:

1. Extract the first character of the string.
2. Convert it to uppercase.
3. Concatenate it with the rest of the string (in lowercase or as-is).

Here’s a simple JavaScript program to capitalize the first letter of a string:

javascript
Copy code
```
function capitalizeFirstLetter(str) {
if (!str || typeof str !== 'string') {
return ''; // Return an empty string if input is invalid
}

// Capitalize the first letter and concatenate with the rest of the string
return str.charAt(0).toUpperCase() + str.slice(1);
}

// Example usage
```

```
let input = "hello world!";
let capitalized = capitalizeFirstLetter(input);

console.log(capitalized); // Output: "Hello world!"
```

Explanation:

1. **charAt(0)**: Extracts the first character of the string.
2. **toUpperCase()**: Converts the first character to uppercase.
3. **slice(1)**: Extracts the rest of the string starting from the second character.
4. **Concatenation**: The uppercase first letter is concatenated with the rest of the string.

Example Output:

For the string "hello world!", the output will be:

Copy code
```
Hello world!
```

This function will capitalize the first letter while keeping the rest of the string unchanged.

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

In JavaScript, you can determine whether a variable is an array using the built-in method Array.isArray(). This method returns true if the variable is an array, and false otherwise.

Here’s a simple JavaScript program to check if a variable is an array:

javascript
Copy code
```
function isArray(variable) {
return Array.isArray(variable);
}

// Example usage
let arrayVar = [1, 2, 3];
let nonArrayVar = "Hello World";

console.log(isArray(arrayVar));     // Output: true
console.log(isArray(nonArrayVar));  // Output: false
```

Explanation:

1. **Array.isArray()**: This method is the recommended way to check if a variable is an array. It returns true for arrays and false for other types of variables.

Example Output:

bash
Copy code
```
true
false
```

This program works for any variable and reliably detects whether it is an array.

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

In JavaScript, you can clone (make a copy of) an array using several methods. The most common ways are:

1. **Using the slice() method**.
2. **Using the spread operator (...)**.
3. **Using the Array.from() method**.

Here’s a program demonstrating how to clone an array using each method:

1. Using slice()

javascript

Copy code

```
function cloneArrayUsingSlice(arr) {
return arr.slice(); // Clones the array
}

// Example usage
let originalArray = [1, 2, 3, 4];
let clonedArray = cloneArrayUsingSlice(originalArray);

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

2. Using the Spread Operator (...)

javascript

Copy code

```
function cloneArrayUsingSpread(arr) {
return [...arr]; // Clones the array using spread
operator
}

// Example usage
let originalArray2 = [5, 6, 7, 8];
let clonedArray2 =
cloneArrayUsingSpread(originalArray2);

console.log(clonedArray2); // Output: [5, 6, 7, 8]
```

3. Using Array.from()

javascript
Copy code

```
function cloneArrayUsingArrayFrom(arr) {
return Array.from(arr); // Clones the array using
Array.from()
}

// Example usage
let originalArray3 = [9, 10, 11, 12];
let clonedArray3 =
cloneArrayUsingArrayFrom(originalArray3);

console.log(clonedArray3); // Output: [9, 10, 11, 12]
```

Explanation:

- **slice()**: Creates a shallow copy of the array by slicing it from the beginning to the end.

- **Spread Operator (...)**: Spreads the elements of the array into a new array, effectively cloning it.
- **Array.from()**: Creates a new array instance from an array-like or iterable object, in this case, an array.

Example Output:

For all examples, the output will be the same:

```
csharp
Copy code
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
```

All of these methods create shallow clones, meaning that if the array contains objects, only the references to those objects are copied.

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

Declaring methods directly within JavaScript objects, while straightforward, can lead to some drawbacks, especially when those methods are used across multiple instances of objects. Below are the key drawbacks:

1. Memory Inefficiency:

When you declare methods directly inside an object, each instance of that object will have its own copy of the method. This increases memory usage unnecessarily, especially if the same method is declared for multiple instances.

Example:

```
javascript
Copy code
function Person(name, age) {
this.name = name;
this.age = age;
this.sayHello = function() { // Method declared
directly in the object
console.log("Hello, my name is " + this.name);
};
}

let person1 = new Person("John", 30);
let person2 = new Person("Jane", 25);

console.log(person1.sayHello === person2.sayHello);
// Output: false (different copies)
```

In this case, each Person object has its own copy of the sayHello method, which consumes more memory.

2. Difficulty in Reusability:

When methods are declared directly in the object, they are tied to that specific object. If you need to reuse the method across different objects or instances, you'll need to declare the method in each one, leading to redundancy and more maintenance complexity.

Example:

```javascript
Copy code
let obj1 = {
name: "Alice",
greet: function() { console.log("Hello " + this.name); }
};

let obj2 = {
name: "Bob",
greet: function() { console.log("Hello " + this.name); }
// Redundant method declaration
};
```

The greet method is declared separately for both objects, which leads to code duplication.

3. Lack of Prototypal Inheritance:

By declaring methods directly in the object, you bypass JavaScript's prototypal inheritance, which allows methods to be shared across all instances. If you place methods on an object's prototype, all instances of the object can share a single copy of the method, improving memory efficiency.

Example:

```
javascript
Copy code
function Person(name, age) {
this.name = name;
this.age = age;
}

Person.prototype.sayHello = function() { // Method
on the prototype
console.log("Hello, my name is " + this.name);
};

let person1 = new Person("John", 30);
let person2 = new Person("Jane", 25);

console.log(person1.sayHello === person2.sayHello);
// Output: true (shared method)
```

In this case, both person1 and person2 share the same sayHello method, improving memory efficiency and code reusability.

4. Performance Considerations:

Declaring methods directly in the object constructor can lead to performance issues, especially when creating many instances. Every time a new object is created, the method has to be redefined, which can slow down execution.

Summary:

- **Memory Inefficiency**: Each object has its own copy of the method, which consumes more memory.
- **Code Duplication**: Direct declaration makes it harder to reuse methods, leading to redundancy.
- **Bypassing Prototypal Inheritance**: Methods aren't shared across instances, missing out on the efficiency and reusability that prototypes provide.
- **Performance Impact**: Declaring methods directly can slow down the creation of multiple objects due to redefinition.

For better efficiency and scalability, it's generally recommended to define methods on the object's prototype, especially when creating multiple instances of the same object.

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

To print the length of a string to the browser console using console.log(), you can follow these simple steps:

1. Define a string variable.
2. Use the .length property of the string to get its length.
3. Print the length using console.log().

Here’s an example:

```
javascript
Copy code
// Define a string
let myString = "Hello, World!";

// Get the length of the string
let stringLength = myString.length;

// Print the length to the browser console
console.log("The length of the string is: " +
stringLength);
```

Explanation:

- **myString.length**: This property returns the number of characters in the string myString.
- **console.log()**: This function prints the specified message to the console.

Output:

If you run the above code in a web browser’s JavaScript console, it will output:

```
csharp
Copy code
The length of the string is: 13
```

This indicates that the string "Hello, World!" contains 13 characters, including spaces and punctuation.

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

To change all the characters in a string to uppercase in JavaScript, you can use the toUpperCase() method. This method converts all the characters in a string to uppercase and returns a new string.

Here’s a simple example demonstrating how to use the toUpperCase() method:

javascript
Copy code
```
// Define a string
let myString = "Hello, World!";

// Convert the string to uppercase
let upperCaseString = myString.toUpperCase();

// Print the uppercase string to the console
console.log("Original String: " + myString);
console.log("Uppercase String: " + upperCaseString);
```
Explanation:

- **toUpperCase()**: This method is called on the myString variable, converting all its characters to uppercase. It does not modify the original string but returns a new string with the changes.
- **console.log()**: This function is used to print both the original and the uppercase string to the console.

Output:

If you run the above code, the output in the console will be:

arduino
Copy code
```
Original String: Hello, World!
Uppercase String: HELLO, WORLD!
```

This shows the original string and its uppercase version.

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

Declaring methods directly in JavaScript objects can lead to several drawbacks, especially concerning memory efficiency, code maintainability, and performance. Here’s a summary of the key drawbacks:

1. Memory Inefficiency:

When you declare methods directly inside an object, each instance of that object has its own copy of the method. This results in increased memory usage, particularly if you create multiple instances of the same object.

Example:

javascript

```
Copy code
function Person(name) {
this.name = name;
this.sayHello = function() { // Method declared
directly in the object
console.log("Hello, my name is " + this.name);
};
}

let person1 = new Person("Alice");
let person2 = new Person("Bob");

console.log(person1.sayHello === person2.sayHello);
// Output: false
```

In this example, person1 and person2 each have their own copy of the sayHello method, consuming more memory than necessary.

2. Code Duplication:

Directly declaring methods in each object leads to code duplication. If the same method is needed in multiple instances, it must be redefined for each one, which increases the chances of errors and complicates maintenance.

Example:

```
javascript
Copy code
let obj1 = {
```

```
name: "Alice",
greet: function() { console.log("Hello " + this.name); }
};

let obj2 = {
name: "Bob",
greet: function() { console.log("Hello " + this.name); }
// Redundant method declaration
};
```

In this case, the greet method is duplicated, leading to more lines of code and potential inconsistencies if the method logic needs to be updated.

3. Bypassing Prototypal Inheritance:

When methods are declared directly within an object, you miss out on JavaScript's prototypal inheritance, which allows methods to be shared across instances. Declaring methods on the object's prototype means that all instances can use the same method without duplicating it.

Example:

javascript
Copy code

```
function Person(name) {
this.name = name;
}
```

```
Person.prototype.sayHello = function() { // Method
on the prototype
console.log("Hello, my name is " + this.name);
};

let person1 = new Person("Alice");
let person2 = new Person("Bob");

console.log(person1.sayHello === person2.sayHello);
// Output: true
```

In this case, both person1 and person2 share the same sayHello method, which improves memory efficiency and reusability.

4. Performance Concerns:

Declaring methods directly inside the object constructor can lead to performance issues, especially when creating many instances. Each time a new object is created, the method has to be redefined, which can slow down execution.

5. Testing and Mocking Difficulties:

If methods are defined directly within an object, it can make testing and mocking those methods more difficult. When using frameworks or libraries for unit testing, it’s often easier to mock methods defined on prototypes.

Summary:

- **Memory Inefficiency**: Each instance has its own copy of methods, leading to increased memory usage.
- **Code Duplication**: Methods must be redefined for each instance, leading to redundancy and maintenance challenges.
- **Bypassing Prototypal Inheritance**: Missing out on the benefits of shared methods through prototypes.
- **Performance Issues**: Slower object creation due to method redefinition.
- **Testing Challenges**: Harder to test and mock methods defined within objects.

For better efficiency and scalability, it is generally recommended to define methods on the object's prototype, especially when creating multiple instances of the same object. This allows for better memory management and easier code maintenance.

Q.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?

To get the current date in JavaScript and format it in various ways such as mm-dd-yyyy, mm/dd/yyyy, dd-mm-yyyy, or dd/mm/yyyy, you can use the Date object along with its methods to extract the month, day, and year. Below is a JavaScript program that demonstrates how to achieve this:

```
javascript
Copy code
function formatCurrentDate() {
// Get the current date
let date = new Date();

// Get month, day, and year
let month = String(date.getMonth() + 1).padStart(2,
'0'); // Months are zero-based
let day = String(date.getDate()).padStart(2, '0');
let year = date.getFullYear();

// Format the date in different styles
let formattedDate1 = `${month}-${day}-${year}`; //
mm-dd-yyyy
let formattedDate2 = `${month}/${day}/${year}`; //
mm/dd/yyyy
let formattedDate3 = `${day}-${month}-${year}`; //
dd-mm-yyyy
let formattedDate4 = `${day}/${month}/${year}`; //
dd/mm/yyyy

// Print the formatted dates
console.log("Formatted Dates:");
console.log(formattedDate1); // mm-dd-yyyy
console.log(formattedDate2); // mm/dd/yyyy
console.log(formattedDate3); // dd-mm-yyyy
console.log(formattedDate4); // dd/mm/yyyy
}

// Call the function
formatCurrentDate();
```

Explanation:

1. **new Date()**: Creates a new Date object containing the current date and time.
2. **getMonth()**: Retrieves the month (0-11). Adding 1 adjusts it to a 1-12 range.
3. **getDate()**: Retrieves the day of the month (1-31).
4. **getFullYear()**: Retrieves the four-digit year.
5. **padStart(2, '0')**: Ensures that month and day are always two digits (e.g., "01" instead of "1").
6. **Template literals** (`` `${...}` ``): Used for easy string interpolation to format the date.

Sample Output:

When you run the program, the output will look something like this (assuming today's date is October 2, 2024):

yaml
Copy code

```
Formatted Dates:
10-02-2024
10/02/2024
02-10-2024
02/10/2024
```

This program provides the current date in various formats as specified.

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

To determine the position of the first occurrence of the letter "a" in the string "30 Days Of JavaScript" using the indexOf method in JavaScript, you can follow these steps:

1. Define the string.
2. Use the indexOf method to find the position of the first occurrence of "a".

Here’s the JavaScript code to accomplish this:

```
javascript
Copy code
// Define the string
let str = "30 Days Of JavaScript";

// Use indexOf to find the position of the first
occurrence of 'a'
let position = str.indexOf('a');

// Print the position
console.log("The position of the first occurrence of
'a':", position);
```

Explanation:

- **indexOf('a')**: This method returns the index of the first occurrence of the specified value ('a' in this case) within the string. If the value is not found, it returns -1.
- **Printing the result**: The position is logged to the console.

Sample Output:

When you run the above code, the output will be:

arduino
Copy code
The position of the first occurrence of 'a': 3

In this case, the letter "a" first appears at index 3 in the string "30 Days Of JavaScript", considering that string indexing starts from 0.

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

To determine the position of the last occurrence of the letter "a" in the string "30 Days Of JavaScript" using the lastIndexOf method in JavaScript, you can follow these steps:

1. Define the string.
2. Use the lastIndexOf method to find the position of the last occurrence of "a".

Here’s the JavaScript code to accomplish this:

```
javascript
Copy code
// Define the string
let str = "30 Days Of JavaScript";

// Use lastIndexOf to find the position of the last
occurrence of 'a'
let lastPosition = str.lastIndexOf('a');

// Print the position
console.log("The position of the last occurrence of
'a':", lastPosition);
```

Explanation:

- **lastIndexOf('a')**: This method returns the index of the last occurrence of the specified value ('a' in this case) within the string. If the value is not found, it returns -1.
- **Printing the result**: The position is logged to the console.

Sample Output:

When you run the above code, the output will be:

```
arduino
Copy code
The position of the last occurrence of 'a': 15
```

In this case, the letter "a" last appears at index 15 in the string "30 Days Of JavaScript", considering that string indexing starts from 0.

Q.54 Form Validtion in JS?

Form validation in JavaScript is crucial for ensuring that the data entered by users is correct and complete before being submitted to a server. Validating forms can enhance user experience by providing immediate feedback and reducing server load.

Here’s a basic example of how to implement form validation using JavaScript:

HTML Form Example

html
Copy code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-
width, initial-scale=1.0">
<title>Form Validation Example</title>
</head>
<body>
<h2>Registration Form</h2>
<form id="registrationForm">
<label for="username">Username:</label>
<input type="text" id="username" name="username"
required>
```

```
<span id="usernameError" style="color:
red;"></span>
<br><br>

<label for="email">Email:</label>
<input type="email" id="email" name="email"
required>
<span id="emailError" style="color: red;"></span>
<br><br>

<label for="password">Password:</label>
<input type="password" id="password"
name="password" required>
<span id="passwordError" style="color:
red;"></span>
<br><br>

<button type="submit">Register</button>
</form>

<script src="formValidation.js"></script>
</body>
</html>
```

JavaScript Validation (formValidation.js)

javascript
Copy code

```
document.getElementById('registrationForm').addEv
entListener('submit', function(event) {
// Prevent form submission
event.preventDefault();
```

```
// Clear previous error messages
document.getElementById('usernameError').textCont
ent = '';
document.getElementById('emailError').textContent
= '';
document.getElementById('passwordError').textCont
ent = '';

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

let isValid = true;

// Username validation
if (username.length < 3) {
document.getElementById('usernameError').textCont
ent = 'Username must be at least 3 characters long.';
isValid = false;
}

// Email validation
const emailPattern =
/^[^\s@]+@[^\s@]+\.[^\s@]+$/; // Simple regex for
email validation
```

```
if (!emailPattern.test(email)) {
document.getElementById('emailError').textContent
= 'Please enter a valid email address.';
isValid = false;
}

// Password validation
if (password.length < 6) {
document.getElementById('passwordError').textCont
ent = 'Password must be at least 6 characters long.';
isValid = false;
}

// If all validations pass, submit the form (or handle
form submission)
if (isValid) {
// Here you can either submit the form or perform
any other actions
alert('Form submitted successfully!');
// Optionally, you can submit the form using:
// this.submit();
}
});
```

Explanation of the Code:

1. **HTML Structure**:
   - A simple registration form is created with fields for username, email, and password. Each input has a corresponding <span> element to display error messages.
2. **JavaScript Logic**:

- An event listener is added to the form for the submit event.
    - event.preventDefault() is used to prevent the default form submission until the validation checks are completed.
    - Previous error messages are cleared at the start of the validation process.
    - Values from the form fields are retrieved, and validation checks are performed:
        - **Username**: Must be at least 3 characters long.
        - **Email**: Checked against a simple regex pattern to ensure it is valid.
        - **Password**: Must be at least 6 characters long.
    - If any validation fails, the appropriate error message is displayed in the corresponding <span> element.
    - If all validations pass, an alert is shown indicating successful submission.

Sample Usage:

You can create a simple HTML file and include the JavaScript code in a separate file named formValidation.js to test the form validation in a web browser.

Additional Considerations:

- **Client-Side Validation**: While client-side validation is useful for user experience, it should

not be solely relied upon. Always perform server-side validation to ensure data integrity.
- **More Complex Validations**: Depending on your requirements, you might want to include more complex validations (e.g., checking for unique usernames, password strength, etc.).
- **Accessibility**: Consider accessibility aspects by ensuring that error messages are screen-reader friendly and visually clear.

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

To create a form with validation for email, number, and password fields in JavaScript, you can follow a structure similar to the previous example. This example will validate the following:

1. **Email**: Must be in a valid email format.
2. **Number**: Must be a valid number and can include a specific range if needed.
3. **Password**: Must meet certain criteria, such as minimum length and complexity.

HTML Form Example

Here's how to create a form with these validations:

html
Copy code
```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-
width, initial-scale=1.0">
<title>Form Validation Example</title>
<style>
span.error {
color: red;
font-size: 0.9em;
}
</style>
</head>
<body>
<h2>Registration Form</h2>
<form id="registrationForm">
<label for="email">Email:</label>
<input type="email" id="email" name="email"
required>
<span id="emailError" class="error"></span>
<br><br>

<label for="number">Number:</label>
<input type="number" id="number" name="number"
required>
<span id="numberError" class="error"></span>
<br><br>

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

```
<input type="password" id="password"
name="password" required>
<span id="passwordError" class="error"></span>
<br><br>

<button type="submit">Register</button>
</form>

<script src="formValidation.js"></script>
</body>
</html>
```

JavaScript Validation (formValidation.js)

Here’s the JavaScript code to validate the form inputs:

javascript
Copy code

```
document.getElementById('registrationForm').addEv
entListener('submit', function(event) {
// Prevent form submission
event.preventDefault();

// Clear previous error messages
document.getElementById('emailError').textContent
= '';
document.getElementById('numberError').textConte
nt = '';
document.getElementById('passwordError').textCont
ent = '';
```

```
}

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

let isValid = true;

// Email validation
const emailPattern =
/^[^\s@]+@[^\s@]+\.[^\s@]+$/; // Simple regex for
email validation
if (!emailPattern.test(email)) {
document.getElementById('emailError').textContent
= 'Please enter a valid email address.';
isValid = false;
}

// Number validation
const num = Number(number);
if (isNaN(num) || num <= 0) {
document.getElementById('numberError').textConte
nt = 'Please enter a valid positive number.';
isValid = false;
}

// Password validation
if (password.length < 6) {
```

```
document.getElementById('passwordError').textCont
ent = 'Password must be at least 6 characters long.';
isValid = false;
}

// If all validations pass, submit the form (or handle
form submission)
if (isValid) {
// Here you can either submit the form or perform
any other actions
alert('Form submitted successfully!');
// Optionally, you can submit the form using:
// this.submit();
}
});
```

Explanation of the Code:

1. **HTML Structure**:
    - The form contains three fields: **Email**, **Number**, and **Password**. Each input has a corresponding <span> element for displaying error messages.
2. **JavaScript Logic**:
    - An event listener is attached to the form to handle the submit event.
    - event.preventDefault() is used to stop the form from submitting until all validations are checked.
    - Previous error messages are cleared before validation begins.

  - The values for email, number, and password are retrieved from the input fields.
  - **Email Validation**: Checks if the email matches a simple regex pattern. If it doesn't, an error message is displayed.
  - **Number Validation**: Checks if the entered value is a valid number and whether it is positive. An error message is displayed if the validation fails.
  - **Password Validation**: Ensures the password is at least 6 characters long, with an error message if it does not meet this requirement.
  - If all validations pass, an alert indicates successful submission. The form can also be submitted programmatically if needed.

Sample Usage:

You can create an HTML file and include the JavaScript code in a separate file named formValidation.js to test the form validation in a web browser.

Additional Considerations:

- **Client-Side vs. Server-Side Validation**: Always perform server-side validation for security, as client-side validation can be bypassed.
- **More Complex Validations**: Depending on your application, you might need additional checks

  (e.g., checking for password strength, validating the number within a specific range, etc.).
- **User Experience**: Consider improving user experience by providing real-time feedback on input fields as the user types.
-
- Q.56 Dynamic Form Validation in JS?

Dynamic form validation in JavaScript allows for real-time feedback as users interact with a form. This can significantly enhance user experience by providing immediate error messages and preventing the submission of invalid data.

Here’s an example of how to implement dynamic form validation in JavaScript using event listeners for input fields.

HTML Form Example

Here’s a simple HTML form with fields for email, number, and password:

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

```
<style>
span.error {
color: red;
font-size: 0.9em;
}
</style>
</head>
<body>
<h2>Dynamic Registration Form</h2>
<form id="registrationForm">
<label for="email">Email:</label>
<input type="email" id="email" name="email"
required>
<span id="emailError" class="error"></span>
<br><br>

<label for="number">Number:</label>
<input type="number" id="number" name="number"
required>
<span id="numberError" class="error"></span>
<br><br>

<label for="password">Password:</label>
<input type="password" id="password"
name="password" required>
<span id="passwordError" class="error"></span>
<br><br>

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

```
<script src="dynamicValidation.js"></script>
</body>
</html>
```

JavaScript Dynamic Validation (dynamicValidation.js)

This script adds event listeners to each input field for real-time validation:

```
javascript
Copy code
document.addEventListener('DOMContentLoaded', ()
=> {
const emailInput =
document.getElementById('email');
const numberInput =
document.getElementById('number');
const passwordInput =
document.getElementById('password');

emailInput.addEventListener('input', validateEmail);
numberInput.addEventListener('input',
validateNumber);
passwordInput.addEventListener('input',
validatePassword);

document.getElementById('registrationForm').addEv
entListener('submit', function(event) {
// Prevent form submission if there are validation
errors
if (!validateEmail() || !validateNumber() ||
!validatePassword()) {
event.preventDefault();
} else {
alert('Form submitted successfully!');
}
});

function validateEmail() {
const emailPattern =
/^[^\s@]+@[^\s@]+\.[^\s@]+$/;
const emailValue = emailInput.value;
const emailError =
document.getElementById('emailError');

if (!emailPattern.test(emailValue)) {
emailError.textContent = 'Please enter a valid email
address.';
return false;
} else {
emailError.textContent = '';
return true;
}
}

function validateNumber() {
const numberValue = Number(numberInput.value);
const numberError =
document.getElementById('numberError');
```

```
if (isNaN(numberValue) || numberValue <= 0) {
numberError.textContent = 'Please enter a valid
positive number.';
return false;
} else {
numberError.textContent = '';
return true;
}
}

function validatePassword() {
const passwordValue = passwordInput.value;
const passwordError =
document.getElementById('passwordError');

if (passwordValue.length < 6) {
passwordError.textContent = 'Password must be at
least 6 characters long.';
return false;
} else {
passwordError.textContent = '';
return true;
}
}
});
```

Explanation of the Code:

1. **HTML Structure**:
   - The form contains fields for **Email**, **Number**, and **Password** with corresponding <span> elements for displaying error messages.

2. **JavaScript Logic**:
    - **Event Listeners**:
        - The DOMContentLoaded event ensures the DOM is fully loaded before the script runs.
        - Event listeners are added to the input fields to validate the inputs on every change (input event).
    - **Validation Functions**:
        - Each input has its own validation function (validateEmail, validateNumber, validatePassword), which checks the input value against specific criteria and updates the error message if validation fails.
    - **Form Submission**:
        - The submit event listener checks all validations before allowing the form to be submitted. If any validation fails, the submission is prevented, and an error message is shown.

Sample Usage:

You can create an HTML file and include the JavaScript code in a separate file named dynamicValidation.js to test the dynamic form validation in a web browser.

Benefits of Dynamic Validation:

- **Immediate Feedback**: Users receive instant feedback on their input, making it easier to correct mistakes.
- **Enhanced User Experience**: Reduces frustration and confusion by guiding users through the input process.
- **Reduced Server Load**: Validating input on the client side can help minimize unnecessary requests to the server with invalid data.

Additional Considerations:

- **Accessibility**: Ensure that error messages are screen-reader friendly and visually clear.
- **Server-Side Validation**: Always validate data on the server side for security reasons, as client-side validation can be bypassed.
- **Custom Validation Logic**: Depending on your needs, you can extend the validation logic to include more complex rules (e.g., matching passwords, ensuring unique usernames, etc.).
-
- Q.57 how many type of JS Event? How to use it ?

JavaScript events are actions or occurrences that happen in the browser, which the JavaScript code can respond to. There are numerous types of JavaScript events, but they can be broadly categorized into several groups. Here's an overview of the main types of JavaScript events and how to use them:

1. Mouse Events

Mouse events are triggered by mouse actions.

- **Common Mouse Events**:
    - click: Triggered when the mouse is clicked.
    - dblclick: Triggered on a double-click.
    - mouseover: Triggered when the mouse pointer is moved over an element.
    - mouseout: Triggered when the mouse pointer moves out of an element.
    - mousedown: Triggered when a mouse button is pressed down.
    - mouseup: Triggered when a mouse button is released.

Example:

javascript
Copy code
```
document.getElementById('myButton').addEventListener('click', function() {
alert('Button was clicked!');
});
```

2. Keyboard Events

Keyboard events are triggered by keyboard actions.

- **Common Keyboard Events**:

    - keydown: Triggered when a key is pressed down.
    - keyup: Triggered when a key is released.
    - keypress: Triggered when a key is pressed down (deprecated; use keydown instead).

Example:

javascript
Copy code
```
document.addEventListener('keydown',
function(event) {
if (event.key === 'Enter') {
alert('Enter key was pressed!');
}
});
```
3. Form Events

Form events are related to user interactions with forms.

- **Common Form Events**:
    - submit: Triggered when a form is submitted.
    - change: Triggered when the value of an element changes.
    - focus: Triggered when an element gains focus.
    - blur: Triggered when an element loses focus.

Example:

javascript

Copy code

```
document.getElementById('myForm').addEventListener('submit', function(event) {
event.preventDefault(); // Prevent form submission
alert('Form submitted!');
});
```

4. Window Events

Window events relate to actions occurring on the window itself.

- **Common Window Events**:
  - load: Triggered when the entire page is loaded.
  - resize: Triggered when the window is resized.
  - scroll: Triggered when the user scrolls the page.
  - unload: Triggered when the user leaves the page.

Example:

javascript
Copy code

```
window.addEventListener('load', function() {
alert('Page is fully loaded!');
});
```

5. Touch Events

Touch events are triggered by touch interactions on touchscreen devices.

- **Common Touch Events**:
    - touchstart: Triggered when a touch point is placed on the touch surface.
    - touchmove: Triggered when a touch point is moved along the touch surface.
    - touchend: Triggered when a touch point is removed from the touch surface.

Example:

javascript
Copy code
```
document.getElementById('myDiv').addEventListener
('touchstart', function() {
alert('Touch started on the div!');
});
```
6. Other Events

There are many other specialized events, including:

- **Custom Events**: You can create your own events using the CustomEvent constructor.
- **Clipboard Events**: Related to clipboard actions, like copy, cut, and paste.

Example:

javascript
Copy code
```
// Create and dispatch a custom event
let myEvent = new CustomEvent('myCustomEvent', {
detail: { key: 'value' } });
document.dispatchEvent(myEvent);
```

```
// Listen for the custom event
document.addEventListener('myCustomEvent',
function(event) {
console.log('Custom event triggered:', event.detail);
});
```

How to Use JavaScript Events

1. **Select the Element**: Use methods like document.getElementById, document.querySelector, or similar to select the DOM element you want to add the event to.
2. **Add an Event Listener**: Use addEventListener to specify the event type and a callback function that will be executed when the event occurs.
3. **Define the Callback Function**: This function will contain the logic to be executed when the event is triggered.

Example of Using Different Events

Here's a complete example that demonstrates different types of events:

html
Copy code

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

```
<title>Event Example</title>
</head>
<body>
<button id="myButton">Click Me!</button>
<input type="text" id="myInput" placeholder="Type
something...">
<form id="myForm">
<input type="submit" value="Submit">
</form>

<script>
// Mouse Event
document.getElementById('myButton').addEventList
ener('click', function() {
alert('Button was clicked!');
});

// Keyboard Event
document.getElementById('myInput').addEventListen
er('keydown', function(event) {
console.log(`Key pressed: ${event.key}`);
});

// Form Event
document.getElementById('myForm').addEventListen
er('submit', function(event) {
event.preventDefault(); // Prevent form submission
alert('Form submitted!');
});
```

```
// Window Event
window.addEventListener('load', function() {
console.log('Page is fully loaded!');
});
</script>
</body>
</html>
```

Summary

- JavaScript supports various events that allow interaction with the user and the environment.
- Different events can be handled using addEventListener, enabling dynamic and responsive web applications.
- Each event type has specific properties and methods associated with it, and event handling can greatly enhance user experience.
-
- Q.60 What is Bom vs Dom in JS?
  In JavaScript, the terms **BOM** (Browser Object Model) and **DOM** (Document Object Model) refer to different parts of the web development ecosystem. Here's a detailed explanation of each, their differences, and their significance:

  ### What is the DOM (Document Object Model)?

  - **Definition**: The DOM is a programming interface that represents the structure of a web page. It provides a way for scripts to manipulate the content, structure, and style of a document.

- **Structure**: The DOM represents the document as a tree of nodes, where each node corresponds to a part of the document (such as elements, attributes, text, etc.). For example, an HTML document can be represented as a tree where each HTML tag is a node.

- **Key Features**:
- Allows dynamic access and updates to the content and structure of web pages.
- Enables event handling and interaction with user input.
- Provides methods and properties to manipulate elements, attributes, and styles.

- **Common Methods**:
- `document.getElementById()`: Accesses an element by its ID.
- `document.querySelector()`: Selects the first element that matches a CSS selector.
- `element.innerHTML`: Gets or sets the HTML content of an element.
- `element.addEventListener()`: Attaches an event handler to an element.

### Example of DOM Manipulation
```javascript
```

// Access an element by ID and change its content
document.getElementById('myElement').innerHTML = 'Hello, World!';
```

### What is the BOM (Browser Object Model)?

- **Definition**: The BOM is a set of objects provided by the browser that allows interaction with the browser window and its components. It is not standardized like the DOM, meaning its implementation can vary between different browsers.

- **Components**: The BOM includes various browser-related objects, such as:
- `window`: Represents the browser window. It is the global object in the browser environment.
- `navigator`: Contains information about the browser (e.g., user agent, online status).
- `location`: Provides information about the current URL and methods to manipulate it.
- `history`: Allows access to the browser's session history.

- **Key Features**:
- Provides methods to manipulate the browser window, such as opening new windows,
```

changing URLs, and controlling the browser history.
- Facilitates features that are not directly related to the document itself.

- **Common Methods**:
- `window.open()`: Opens a new browser window or tab.
- `window.alert()`: Displays an alert dialog box.
- `location.href`: Gets or sets the current URL of the browser.
- `history.back()`: Navigates to the previous page in the session history.

### Example of BOM Manipulation
```javascript
// Redirect to another URL
window.location.href =
'https://www.example.com';
```

### Key Differences Between BOM and DOM

Feature	BOM (Browser Object Model)	DOM (Document Object Model)
Definition	A set of objects for interacting with the browser window and its components.	A representation of the

document structure (HTML or XML) as a tree of nodes. |
Focus	Browser-related features and functionality.	Document content, structure, and presentation.
Objects	Includes `window`, `navigator`, `location`, and `history`.	Includes elements, attributes, and text nodes in a document.
Usage	Used for browser control and navigation.	Used for manipulating document content and structure.
Standardization	Not standardized across all browsers; varies between implementations.	Standardized by the W3C and is consistent across compliant browsers.

### Summary
- **DOM** is used to manipulate the content and structure of a web page, while **BOM** provides methods for interacting with the browser itself.
- Both BOM and DOM are integral to web development, allowing developers to create interactive and dynamic web applications. Understanding the differences between them helps in effectively utilizing JavaScript in a web environment.

Q.61 Array vs object defences in JS?
In JavaScript, **arrays** and **objects** are both fundamental data structures used to store collections of data, but they have different characteristics and uses. Here's a breakdown of their definitions, differences, and when to use each.

### What is an Array?

- **Definition**: An array is a special type of object that is used to store an ordered collection of values. Arrays can hold any type of data, including numbers, strings, objects, and even other arrays.

- **Characteristics**:
- **Ordered**: The elements in an array have a specific order, and each element can be accessed using its index (starting from 0).
- **Zero-Based Indexing**: The first element is at index 0, the second at index 1, and so on.
- **Dynamic Size**: Arrays can grow and shrink in size as needed. You can add or remove elements at any time.

- **Common Methods**:
- `push()`: Adds an element to the end of the array.
- `pop()`: Removes the last element from the array.
- `shift()`: Removes the first element from the array.
- `unshift()`: Adds an element to the beginning of the array.

- `map()`, `filter()`, `reduce()`: Functional programming methods for transforming and processing arrays.

- **Example**:
```javascript
let fruits = ['apple', 'banana', 'orange'];
console.log(fruits[1]); // Output: banana
fruits.push('grape'); // Adds 'grape' to the end of the array
```

### What is an Object?

- **Definition**: An object is a collection of key-value pairs. Each key is a string (also called a property name), and the value can be any type of data, including other objects, arrays, functions, etc.

- **Characteristics**:
- **Unordered**: The properties in an object do not have a specific order, and they are accessed using their keys (property names) rather than indices.
- **Dynamic**: Objects can have properties added, modified, or removed at any time.
- **Key-Value Structure**: Each property has a unique key associated with a value.

- **Common Methods**:
- `Object.keys()`: Returns an array of the object's

keys.
- `Object.values()`: Returns an array of the object's values.
- `Object.entries()`: Returns an array of the object's key-value pairs.
- `hasOwnProperty()`: Checks if the object has a specific property.

- **Example**:
```javascript
let person = {
name: 'Alice',
age: 30,
hobbies: ['reading', 'traveling'],
greet: function() {
console.log('Hello, ' + this.name);
}
};
console.log(person.name); // Output: Alice
person.greet(); // Output: Hello, Alice
```

### Key Differences Between Arrays and Objects

Feature	Arrays	Objects
Structure	Ordered collection of elements.	Unordered collection of key-

value pairs. |
Access Method	Accessed by index (numerical).	Accessed by keys (string).
Use Case	Ideal for lists of items (e.g., numbers, strings).	Ideal for representing entities with properties (e.g., users, products).
Iteration	Commonly used with loop methods (e.g., `forEach`, `map`).	Commonly iterated using `for...in` or `Object.keys()`.
Length Property	Has a `length` property that returns the number of elements.	No inherent length; size must be determined manually.
Methods	Includes methods for manipulating ordered data (e.g., `push`, `pop`).	Includes methods for manipulating properties (e.g., `Object.keys`, `delete`).

### When to Use Arrays vs. Objects

- **Use Arrays** when:
- You need to maintain an ordered list of items.
- You want to perform operations that involve the order of elements (e.g., sorting).
- You want to utilize built-in array methods for transformation and manipulation.

- **Use Objects** when:
- You want to represent complex data structures with properties.

- You need to store related data as key-value pairs.
- You want to model entities with specific attributes (like a user profile).

### Summary
- **Arrays** are designed for ordered collections of data and provide convenient methods for manipulation.
- **Objects** are intended for storing related data as key-value pairs, making them ideal for representing complex entities.
- Understanding the differences between arrays and objects helps in choosing the right data structure for your specific use case in JavaScript development.

Q.62 Split the string into an array using split() Method?

In JavaScript, you can split a string into an array of substrings using the `split()` method. This method divides a string into an array of substrings based on a specified delimiter (also called a separator). Here's how to use the `split()` method effectively.

### Syntax

```javascript
string.split(separator, limit);
```

- **separator**: The string or regular expression

that defines the points at which to split the string. If omitted, the entire string will be returned as a single element array.
- **limit** (optional): An integer that specifies a limit on the number of splits to be found. The `split()` method will split the string at each occurrence of the separator until the limit is reached.

### Examples

1. **Splitting a String by Space**:
```javascript
const sentence = "Hello world, this is JavaScript!";
const words = sentence.split(" ");
console.log(words);
// Output: ["Hello", "world,", "this", "is", "JavaScript!"]
```

2. **Splitting a String by Comma**:
```javascript
const csv = "apple,banana,cherry,dates";
const fruits = csv.split(",");
console.log(fruits);
// Output: ["apple", "banana", "cherry", "dates"]
```

3. **Splitting a String with a Limit**:
```javascript
```

const sentence = "One,Two,Three,Four,Five";
const limitedSplit = sentence.split(",", 3);
console.log(limitedSplit);
// Output: ["One", "Two", "Three"]

4. **Splitting a String by Regular Expression**:
```javascript
const data = "one;two|three,four";
const splitData = data.split(/[,;|]/);
console.log(splitData);
// Output: ["one", "two", "three", "four"]
```

5. **Splitting a String into Characters**:
```javascript
const str = "Hello";
const chars = str.split("");
console.log(chars);
// Output: ["H", "e", "l", "l", "o"]
```

### Summary
The `split()` method is a versatile tool for breaking strings into arrays based on specific criteria. It allows for various delimiters and can be customized with an optional limit on the number of splits, making it a valuable function for string manipulation in JavaScript.

Q.63 Check if the string contains a word Script using includes() method?

In JavaScript, you can split a string into an array of substrings using the `split()` method. This method divides a string into an array of substrings based on a specified delimiter (also called a separator). Here’s how to use the `split()` method effectively.

### Syntax

```javascript
string.split(separator, limit);
```

- **separator**: The string or regular expression that defines the points at which to split the string. If omitted, the entire string will be returned as a single element array.
- **limit** (optional): An integer that specifies a limit on the number of splits to be found. The `split()` method will split the string at each occurrence of the separator until the limit is reached.

### Examples

1. **Splitting a String by Space**:
```javascript
const sentence = "Hello world, this is JavaScript!";
const words = sentence.split(" ");
```

console.log(words);
// Output: ["Hello", "world,", "this", "is", "JavaScript!"]

2. **Splitting a String by Comma**:
```javascript
const csv = "apple,banana,cherry,dates";
const fruits = csv.split(",");
console.log(fruits);
// Output: ["apple", "banana", "cherry", "dates"]
```

3. **Splitting a String with a Limit**:
```javascript
const sentence = "One,Two,Three,Four,Five";
const limitedSplit = sentence.split(",", 3);
console.log(limitedSplit);
// Output: ["One", "Two", "Three"]
```

4. **Splitting a String by Regular Expression**:
```javascript
const data = "one;two|three,four";
const splitData = data.split(/[,;|]/);
console.log(splitData);
// Output: ["one", "two", "three", "four"]
```

5. **Splitting a String into Characters**:

```javascript
const str = "Hello";
const chars = str.split("");
console.log(chars);
// Output: ["H", "e", "l", "l", "o"]
```

### Summary
The `split()` method is a versatile tool for breaking strings into arrays based on specific criteria. It allows for various delimiters and can be customized with an optional limit on the number of splits, making it a valuable function for string manipulation in JavaScript.

Q.63 Check if the string contains a word Script using includes() method?
In JavaScript, you can check if a string contains a specific substring (such as the word "Script") using the `includes()` method. This method returns `true` if the substring is found within the string, and `false` otherwise.

### Syntax

```javascript
string.includes(substring, start);
```

- **substring**: The string to search for.

- **start** (optional): The position in the string to start the search. The default is 0.

### Example

Here’s how to use the `includes()` method to check for the presence of the word "Script":

1. **Basic Example**:
```javascript
const text = "JavaScript is a versatile programming language.";
const containsScript = text.includes("Script");
console.log(containsScript);
// Output: true
```

2. **Case Sensitivity**:
The `includes()` method is case-sensitive. Therefore, checking for "script" (lowercase) will return `false`.
```javascript
const text = "JavaScript is a versatile programming language.";
const containsScriptLower = text.includes("script");
console.log(containsScriptLower);
// Output: false
```

3. **Using with User Input**:
```javascript
```

const userInput = "I love learning about JavaScript
and its frameworks.";
const hasScript = userInput.includes("Script");
console.log(hasScript);
// Output: true

4. **Checking from a Specific Position**:
You can also specify a starting position for the
search:
```javascript
const text = "Learning JavaScript is fun.";
const containsScriptFromPosition =
text.includes("Script", 10); // Start checking from
index 10
console.log(containsScriptFromPosition);
// Output: true
```

### Summary
The `includes()` method is a convenient way to
determine if a string contains a specific substring. It
is case-sensitive and can also start searching from a
specified index, making it versatile for various string-
checking scenarios.

Q.64 Change all the string characters to lowercase
letters using toLowerCase() Method.
In JavaScript, you can change all the characters in a
string to lowercase using the `toLowerCase()`

method. This method returns a new string with all the characters converted to lowercase without modifying the original string.

### Syntax

```javascript
string.toLowerCase();
```

### Example

Here’s how to use the `toLowerCase()` method:

1. **Basic Example**:
```javascript
const originalString = "Hello, World!";
const lowerCaseString =
originalString.toLowerCase();
console.log(lowerCaseString);
// Output: "hello, world!"
```

2. **Using with User Input**:
```javascript
const userInput = "JAVASCRIPT is FUN!";
const lowerCaseInput = userInput.toLowerCase();
console.log(lowerCaseInput);
// Output: "javascript is fun!"
```

3. **Changing All Characters in a String**:
```javascript
const mixedCaseString = "ThIs Is A mIXed CASE
StrIng.";
const allLowerCase =
mixedCaseString.toLowerCase();
console.log(allLowerCase);
// Output: "this is a mixed case string."
```

### Summary
The `toLowerCase()` method is straightforward and effective for converting any string to lowercase. It is especially useful for case-insensitive comparisons and formatting strings consistently.

Q.65 What is Character at index 15 in ’30 Days of JavaScript’ string? Use charAt() method.
In JavaScript, you can use the `charAt()` method to get the character at a specific index in a string. The method takes an integer argument that represents the index of the character you want to retrieve.

### Syntax

```javascript
string.charAt(index);
```

- **index**: The position of the character you want to retrieve (starting from 0).

### Example

To find the character at index 15 in the string `'30 Days of JavaScript'`, you can use the following code:

```javascript
const str = '30 Days of JavaScript';
const characterAtIndex15 = str.charAt(15);
console.log(characterAtIndex15);
// Output: 'S'
```

### Explanation

In this example:
- The string `'30 Days of JavaScript'` has the following characters indexed as shown below:

```
Index:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
17 18
Chars: '3', '0', ' ', 'D', 'a', 'y', 's', ' ', 'o', 'f', ' ', 'J', 'a', 'v',
'a', 'S', 'c', 'r', 'i', 'p', 't'
```

- The character at index 15 is **'S'**.

### Summary
The `charAt()` method is an easy way to access individual characters in a string by their index. In this case, the character at index 15 in `'30 Days of JavaScript'` is **'S'**.

Q.66 copy to one string to another string in JS?
In JavaScript, you can copy the contents of one string to another string simply by assigning the first string to the second string variable. Strings are immutable in JavaScript, so you can't change the original string directly, but you can create a new reference to it.

### Example

Here’s how to copy one string to another:

```javascript
// Original string
let originalString = "Hello, World!";

// Copying the string to another string
let copiedString = originalString;

// Displaying the strings
console.log("Original String: ", originalString);  //
Output: Hello, World!
console.log("Copied String: ", copiedString);      //
Output: Hello, World!
```

### Explanation

- **String Assignment**: The line `let copiedString = originalString;` creates a new reference `copiedString` that points to the same string value as `originalString`.
- **Immutability**: Since strings in JavaScript are immutable, any changes made to `copiedString` will not affect `originalString`, and vice versa.

### Modifying the Copied String

If you want to modify the copied string, you can do so without affecting the original string:

```javascript
// Modifying the copied string
copiedString = "Goodbye, World!";

// Displaying the modified copied string and the original string
console.log("Modified Copied String: ", copiedString);  // Output: Goodbye, World!
console.log("Original String After Modification: ", originalString);  // Output: Hello, World!
```

### Summary

Copying one string to another in JavaScript is as simple as assigning it to a new variable. You can freely manipulate the copied string without affecting the original string, thanks to the immutability of string values.