Example 1: Check Palindrome Using for Loop

```
// program to check if the string is palindrome or not

function checkPalindrome(string) {

    // find the length of a string
    const len = string.length;

    // loop through half of the string
    for (let i = 0; i < len / 2; i++) {

        // check if first and last string are same
        if (string[i] !== string[len - 1 - i]) {
            return 'It is not a palindrome';
        }
    }
    return 'It is a palindrome';
}

// take input
const string = prompt('Enter a string: ');

// call the function
const value = checkPalindrome(string);

console.log(value);
```

Run Code

Output

```
Enter a string: madam
It is a palindrome
```

Example 2: Check Palindrome using built-in Functions

```
// program to check if the string is palindrome or not

function checkPalindrome(string) {

    // convert string to an array
    const arrayValues = string.split('');

    // reverse the array values
    const reverseArrayValues = arrayValues.reverse();
```

```
    // convert array to string
    const reverseString = reverseArrayValues.join('');

    if(string == reverseString) {
        console.log('It is a palindrome');
    }
    else {
        console.log('It is not a palindrome');
    }
}

//take input
const string = prompt('Enter a string: ');

checkPalindrome(string);
```

Run Code

Output

```
Enter a string: hello
It is not a palindrome
```

In the above program, the palindrome is checked using the built-in methods available in JavaScript.

- The `split('')` method converts the string into individual array characters.

```
const arrayValues = string.split(''); // ["h", "e", "l", "l", "o"]
```

- The `reverse()` method reverses the position in an array.

```
// ["o", "l", "l", "e", "h"]
```

```
const reverseArrayValues = arrayValues.reverse();
```

- The `join('')` method joins all the elements of an array into a string.

```
const reverseString = reverseArrayValues.join(''); // "olleh"
```

- Then the `if...else` statement is used to check if the string and the reversed string are equal. If they are equal, the string is a palindrome.

Note: The multiple lines of code can be reduced and written in one line:

```
const reverseString = string.split('').reverse().join('');
```

Example: Sort Words in Alphabetical Order

```
// program to sort words in alphabetical order

// take input
const string = prompt('Enter a sentence: ');

// converting to an array
const words = string.split(' ');

// sort the array elements
words.sort();

// display the sorted words
console.log('The sorted words are:');

for (const element of words) {
  console.log(element);
}
```

Run Code

Output

```
Enter a sentence: I am learning JavaScript
The sorted words are:
I
JavaScript
am
learning
```

Why I and JavaScript are printed before am?

This is because `I` and `J` of `JavaScript` are in uppercase. And, when we use the `sort()` method, uppercase letters are placed before lowercase.

We can verify this by providing only lowercase input.

```
// program to sort words in alphabetical order

// take input
const string = prompt('Enter a sentence: ');

// converting to an array
const words = string.split(' ');

// sort the array elements
words.sort();
```

```
// display the sorted words
console.log('The sorted words are:');

for (const element of words) {
  console.log(element);
}
```
Run Code

Output

```
Enter a sentence: i am learning javascript
The sorted words are:
am
i
javascript
learning
```

Here, we get the expected output now.

Note: Instead of displaying from the array values, you can also convert the array elements back to the string and display the values as a string using `join()` method.

```
words.join(' '); // I JavaScript am learning
```

Example: Replace First Occurrence of a Character in a String

```
// program to replace a character of a string

const string = 'Mr Red has a red house and a red car';

// replace the characters
const newText = string.replace('red', 'blue');

// display the result
console.log(newText);
```
Run Code

Output

```
Mr Red has a blue house and a red car
```

Example 2: Replace Character of a String Using RegEx

```
// program to replace a character of a string

const string = 'Mr Red has a red house and a red car';

// regex expression
const regex = /red/g;

// replace the characters
const newText = string.replace(regex, 'blue');

// display the result
console.log(newText);
```

Run Code

Output

```
Mr Red has a blue house and a blue car
```

Example 1: Reverse a String Using for Loop

```
// program to reverse a string

function reverseString(str) {

    // empty string
    let newString = "";
    for (let i = str.length - 1; i >= 0; i--) {
        newString += str[i];
    }
    return newString;
}

// take input from the user
const string = prompt('Enter a string: ');

const result = reverseString(string);
console.log(result);
```

Run Code

Output

```
Enter a string: hello world
dlrow olleh
```

Example 2: Reverse a String Using built-in Methods

```
// program to reverse a string

function reverseString(str) {

    // return a new array of strings
    const arrayStrings = str.split("");

    // reverse the new created array elements
    const reverseArray = arrayStrings.reverse();

    // join all elements of the array into a string
    const joinArray = reverseArray.join("");

    // return the reversed string
    return joinArray;
}

// take input from the user
const string = prompt('Enter a string: ');

const result = reverseString(string);
console.log(result);
```

Run Code

Output

```
Enter a string: hello
olleh
```

Example 1: Check Occurrence of a Character Using for Loop

```
// program to check the number of occurrence of a character

function countString(str, letter) {
    let count = 0;

    // looping through the items
    for (let i = 0; i < str.length; i++) {

        // check if the character is at that position
        if (str.charAt(i) == letter) {
            count += 1;
        }
```

```
    }
    return count;
}

// take input from the user
const string = prompt('Enter a string: ');
const letterToCheck = prompt('Enter a letter to check: ');

//passing parameters and calling the function
const result = countString(string, letterToCheck);

// displaying the result
console.log(result);
```
Run Code

Output

```
Enter a string: school
Enter a  letter to check: o
2
```

Example 2: Check occurrence of a character using a Regex

```
// program to check the occurrence of a character

function countString(str, letter) {

    // creating regex
    const re = new RegExp(letter, 'g');

    // matching the pattern
    const count = str.match(re).length;

    return count;
}

// take input from the user
const string = prompt('Enter a string: ');
const letterToCheck = prompt('Enter a letter to check: ');

//passing parameters and calling the function
const result = countString(string, letterToCheck);

// displaying the result
console.log(result);
```
Run Code

Output

```
Enter a string: school
Enter a  letter to check: o
2
```

Example 1: Convert First letter to UpperCase

```
// program to convert first letter of a string to uppercase
function capitalizeFirstLetter(str) {

    // converting first letter to uppercase
    const capitalized = str.charAt(0).toUpperCase() + str.slice(1);

    return capitalized;
}

// take input
const string = prompt('Enter a string: ');

const result = capitalizeFirstLetter(string);

console.log(result);
```

Run Code

Output

```
Enter a string: javaScript
JavaScript
```

In the above program, the user is prompted to enter a string and that string is passed into the `capitalizeFirstLetter()` function.

- The string's first character is extracted using `charAt()` method. Here, `str.charAt(0);` gives `j`.
- The `toUpperCase()` method converts the string to uppercase.
  Here, `str.charAt(0).toUpperCase();` gives `J`.
- The `slice()` method returns the rest of the string.
  Here, `str.slice(1);` gives `avaScript`.
- These two values are concatenated using the `+` operator.

Note: You can also extract the first character of a string using an array accessing property: `str[0]`.

```
str.str[0]; // j
```

Example 2: Convert First letter to UpperCase using Regex

```
// program to convert first letter of a string to uppercase
function capitalizeFirstLetter(str) {

    // converting first letter to uppercase
    const capitalized = str.replace(/^./, str[0].toUpperCase());

    return capitalized;
}

// take input
const string = prompt('Enter a string: ');

const result = capitalizeFirstLetter(string);

console.log(result);
```

Run Code

Output

```
Enter a string: javaScript
JavaScript
```

In the above program, the regular expression (regex) is used to convert the first letter of a string to uppercase.

- The regex pattern is `/^./` matches the first character of a string.
- The `toUpperCase()` method converts the string to uppercase.

The five letters **a, e, i, o** and **u** are called vowels. All other alphabets except these **5** vowels are called consonants.

Example 1: Count the Number of Vowels Using Regex

```
// program to count the number of vowels in a string

function countVowel(str) {

    // find the count of vowels
```

```
    const count = str.match(/[aeiou]/gi).length;

    // return number of vowels
    return count;
}

// take input
const string = prompt('Enter a string: ');

const result = countVowel(string);

console.log(result);
```
Run Code

Output

```
Enter a string: JavaScript program
5
```

In the above program, the user is prompted to enter a string and that string is passed to the `countVowel()` function.

- The regular expression (RegEx) pattern is used with the `match()` method to find the number of vowels in a string.
- The pattern `/[aeiou]/gi` checks for all the vowels (case-insensitive) in a string. Here, `str.match(/[aeiou]/gi);` gives `["a", "a", "i", "o", "a"]`
- The `length` property gives the number of vowels present.

Example 2: Count the Number of Vowels Using for Loop

```
// program to count the number of vowels in a string

// defining vowels
const vowels = ["a", "e", "i", "o", "u"]

function countVowel(str) {
    // initialize count
    let count = 0;

    // loop through string to test if each character is a vowel
    for (let letter of str.toLowerCase()) {
        if (vowels.includes(letter)) {
            count++;
        }
```

```
    }

    // return number of vowels
    return count
}

// take input
const string = prompt('Enter a string: ');

const result = countVowel(string);

console.log(result);
```
Run Code

Output

```
Enter a string: JavaScript program
5
```

In the above example,

- All the vowels are stored in a `vowels` array.
- Initially, the value of the `count` variable is **0**.
- The `for...of` loop is used to iterate over all the characters of the string.
- The `toLowerCase()` method converts all the characters of a string to lowercase.
- The `includes()` method checks if the `vowel` array contains any of the characters of the string.
- If any character matches, the value of `count` is increased by **1**.

JavaScript Program to Check Whether a String Starts and Ends With Certain Characters

Example 1: Check String Using Built-in Methods

```
// program to check if a string starts with 'S' and ends with 'G'

function checkString(str) {

    // check if the string starts with S and ends with G
    if(str.startsWith('S') && str.endsWith('G')) {
        console.log('The string starts with S and ends with G');
    }
    else if(str.startsWith('S')) {
        console.log('The string starts with S but does not end with G');
    }
```

```
     else if(str.endsWith('G')) {
        console.log('The string starts does not with S but end with G');
    }
    else {
        console.log('The string does not start with S and does not end with G');
    }
}


// take input
let string = prompt('Enter a string: ');
checkString(string);
```

Run Code

Output

```
Enter a string: String
The string starts with S but does not end with G
```

In the above program, the two methods `startsWith()` and `endsWith()` are used.

- The `startsWith()` method checks if the string starts with the particular string.
- The `endsWith()` method checks if the string ends with the particular string.

The above program does not check for lowercase letters. Hence, here **G** and **g** are different.

You could also check if the above character starts with **S** or **s** and ends with **G** or **g**.

```
str.startsWith('S') || str.startsWith('s') && str.endsWith('G') || str.endsWith('g');
```

Example 2: Check The String Using Regex

```
// program to check if a string starts with 'S' and ends with 'G'

function checkString(str) {

    // check if the string starts with S and ends with G
    if( /^S/i.test(str) && /G$/i.test(str)) {
        console.log('The string starts with S and ends with G');
    }
    else if(/^S/i.test(str)) {
        console.log('The string starts with S but does not ends with G');
    }
     else if(/G$/i.test(str)) {
        console.log('The string starts does not with S but ends with G');
```

```
    }
    else {
        console.log('The string does not start with S and does not end with G');
    }
}

// for loop to show different scenario
for (let i = 0; i < 3; i++) {

    // take input
    const string = prompt('Enter a string: ');

    checkString(string);
}
```
Run Code

Output

```
Enter a string: String
The string starts with S and ends with G
Enter a string: string
The string starts with S and ends with G
Enter a string: JavaScript
The string does not start with S and does not end with G
```

In the above program, a regular expression (RegEx) is used with the `test()` method to check if the string starts with **S** and ends with **G**.

- The `/^S/i` pattern checks if the string is **S** or **s**. Here, `i` denotes that the string is case-insensitive. Hence, **S** and **s** are considered the same.
- The `/G$/i` patterns checks if the string is **G** or **g**.
- The `if...else...if` statement is used to check the conditions and display the outcome accordingly.
- The `for` loop is used to take different inputs from the user to show different cases.

Example 1: Replace All Occurrence of String Using RegEx

```
// program to replace all occurrence of a string

const string = 'Mr Red has a red house and a red car';

// regex expression
const regex = /red/gi;

// replace the characters
```

```
const newText = string.replace(regex, 'blue');

// display the result
console.log(newText);
```
Run Code

Output

```
Mr blue has a blue house and a blue car
```

In the above program, a regex expression is used as the first parameter inside the `replace()` method.

`/g` refers to global (that replacement is done across the whole string) and `/i` refers to case-insensitive.

The `replace()` method takes the string that you want to replace as the first parameter and the string you want to replace with as the second parameter.

Example 2: Replace All Occurrence of String Using built-in Method

```
// program to replace all occurrence of a string

const string = 'Mr red has a red house and a red car';

const result = string.split('red').join('blue');

console.log(result);
```
Run Code

Output

```
Mr blue has a blue house and a blue car
```

In the above program, the built-in `split()` and `join()` method is used to replace all the occurrences of the string.

- The string is split into individual array elements using the `split()` method.
  Here, `string.split('red')` gives `["Mr ", " has a ", " house and a ", " car"]` by splitting the string.

- The array elements are joined into a single string using the `join()` method.
  Here, `reverseArray.join('blue')` gives `Mr blue has a blue house and a blue car` by joining the array elements.

Example 1: Create Multiline Strings Using +

```
// program to create a multiline strings

// using the + operator
const message = 'This is a long message\n' +
    'that spans across multiple lines\n' +
    'in the code.'

console.log(message);
```

Run Code

Output

```
This is a long message
that spans across multiple lines
in the code.
```

In the above example, a multiline string is created using the `+` operator and `\n`.
The escape character `\n` is used to break the line.

Example 2: Create Multiline Strings Using \

```
// program to create a multiline strings

// using the \ operator
const message = 'This is a long message\n \
that spans across multiple lines\n \
in the code.'

console.log(message);
```

Run Code

Output

```
This is a long message
that spans across multiple lines
in the code.
```

In the above example, a multiline string is created using `\`. `\n` is used to break the line.

Example 3: Create Multiline Strings Using Template Literal

```
// program to create a multiline strings

// using the template literal

const message = `This is a long message
that spans across multiple lines
in the code.`

console.log(message);
```

Run Code

Output

```
This is a long message
that spans across multiple lines
in the code.
```

In the above example, the template literal `` ` ` `` is used to write multiline strings.

The template literal was introduced in the newer version of JavaScript (**ES6**).

Example 1: Format Numbers as Currency String

```
// program to format numbers as currency string
const formatter = new Intl.NumberFormat('en-US', {
    style: 'currency',
    currency: 'USD'
});

formatter.format(2500);
```

Run Code

Output

```
$2,500.00
```

In the above program, we have used the `Intl.NumberFormat` object.

The `Intl.NumberFormat` object enables language-sensitive number formatting.

Example 2: Format Numbers as Currency String Using concatenation

```
// program to format numbers as currency string

const number = 1234.5678;

const result = '$ ' + number.toFixed(2);

console.log(result);
```

Run Code

Output

```
$ 1234.57
```

In the above example, the `toFixed(2)` method is used to round up the number to two decimal values.

`'$'` is added to the number to convert it into a currency string.

Example 3: Format Numbers as Currency String Using toLocaleString()

```
// program to format numbers as currency string

const result =  (2500).toLocaleString('en-US', {
    style: 'currency',
    currency: 'USD'
});

console.log(result);
```

Run Code

Output

```
$2,500.00
```

The `toLocaleString()` method returns a string with a language-sensitive representation of that number.

Example 4: Format Numbers as Currency String Using RegEx

```
// program to format numbers as currency string

const result = 1234.5678.toFixed(2).replace(/\d(?=(\d{3})+\.)/g, '$&,');

console.warn('$ ' + result);
```

Run Code

Output

```
$ 1,234.57
```

In the above example, the `replace()` method is used with the RegEx pattern to replace the number to currency string.

The `toFixed(2)` method is used to round up the number to two decimal values.

Example 1: Generate Random Strings

```
// program to generate random strings

// declare all characters
const characters ='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

function generateString(length) {
    let result = ' ';
    const charactersLength = characters.length;
    for ( let i = 0; i < length; i++ ) {
        result += characters.charAt(Math.floor(Math.random() * charactersLength));
    }

    return result;
}
```

```
console.log(generateString(5));
Run Code
```

Output

```
B5cgH
```

In the above example, the `Math.random()` method is used to generate random characters from the specified characters **(A-Z, a-z, 0-9)**.

The `for` loop is used to loop through the number passed into the `generateString()` function. During each iteration, a random character is generated.

Example 2: Generate Random Strings Using Built-in Methods

```
// program to generate random strings

const result = Math.random().toString(36).substring(2,7);
console.log(result);
Run Code
```

Output

```
gyjvo
```

In the above example, built-in methods are used to generate random characters.

The `Math.random()` method generates the random number between **0** and **1**.

In `toString(36)` method, **36** represents **base 36**. The `toString(36)` represents digits beyond 9 by letters.

The `substring(2, 7)` method returns five characters.

> **Note**: In the above examples, the output varies each time because random characters are generated at every execution.

Example 1: Using startsWith()

```
// program to check if a string starts with another string
```

```
const string = 'hello world';

const toCheckString = 'he';

if(string.startsWith(toCheckString)) {
    console.warn('The string starts with "he".');
}
else {
    console.warn(`The string does not starts with "he".`);
}
```
Run Code

Output

```
The string starts with "he".
```

In the above program, the `startsWith()` method is used to determine if the string starts with **'he'**. The `startsWith()` method checks if the string starts with the particular string. The `if...else` statement is used to check the condition.

Example 2: Using lastIndexOf()

```
// program to check if a string starts with another string

const string = 'hello world';

const toCheckString = 'he';

let result = string.lastIndexOf(toCheckString, 0) === 0;
if(result) {
    console.warn('The string starts with "he".');
}
else {
    console.warn(`The string does not starts with "he".`);
}
```
Run Code

Output

```
The string starts with "he".
```

In the above program, the `lastIndexOf()` method is used to check if a string starts with another string.

The `lastIndexOf()` method returns the index of the searched string (here searching from the first index).

Example 3: Using RegEx

```
// program to check if a string starts with another string

const string = 'hello world';

const pattern = /^he/;

let result = pattern.test(string);

if(result) {
    console.warn('The string starts with "he".');
}
else {
    console.warn(`The string does not starts with "he".`);
}
```

Run Code

Output

```
The string starts with "he".
```

In the above program, the string is checked using the RegEx pattern and the `test()` method. `/^` indicates the starting of the string.

Example 1: Trim a String

```
// program to trim a string

const string = '      Hello World       ';

const result = string.trim();

console.log(result);
```

Run Code

Output

```
Hello World
```

In the above example, the `trim()` method is used to trim a string.

The `trim()` method removes white space from both sides of the string.

Example 2: Trim a String Using RegEx

```
// program to trim a string

function trimString(x) {
    let trimValue = x.replace(/^\s+|\s+$/g,'');
    return trimValue;
}

const result = trimString('    Hello world    ');
console.log(result);
```

Run Code

Output

```
Hello World
```

In the above program, the RegEx is used with the `replace()` method to trim the string.

`/^\s+|\s+$/g` checks for whitespace at the beginning and end of the string.

JavaScript Program to Check Whether a String Contains a Substring

Example 1: Check String with includes()

```
// program to check if a string contains a substring

// take input
const str = prompt('Enter a string:');
const checkString = prompt('Enter a string that you want to check:');
```

```
// check if string contains a substring
if(str.includes(checkString)) {
    console.log(`The string contains ${checkString}`);
} else {
    console.log(`The string does not contain ${checkString}`);
}
```
Run Code

Output

```
Enter a string: JavaScript is fun
Enter a string that you want to check: fun
The string contains fun
```

The `includes()` method is used with the `if...else` statement to check whether a string contains the characters of a specified string.

> **Note**: The `includes()` method is case-sensitive. Hence, **fun** and **Fun** are different.

Example 2: Check String with indexOf()

```
// program to check if a string contains a substring

// take input
const str = prompt('Enter a string:');
const checkString = prompt('Enter a string that you want to check:');

// check if string contains a substring
if(str.indexOf(checkString) !== -1) {
    console.log(`The string contains ${checkString}`);
} else {
    console.log(`The string does not contain ${checkString}`);
}
```
Run Code

Output

```
Enter a string: JavaScript is fun
Enter a string that you want to check: fun
The string contains fun
```

In the above program, the `indexOf()` method is used with the `if...else` statement to check if a string contains a substring.

The `indexOf()` method searches a string and returns the position of the first occurrence. When a substring cannot be found, it returns **-1**.

> **Note**: The `indexOf()` method is case sensitive.

JavaScript Program to Compare Two Strings

Example 1: Using toUpperCase()

```
// js program to perform string comparison

const string1 = 'JavaScript Program';
const string2 = 'javascript program';

// compare both strings
const result = string1.toUpperCase() === string2.toUpperCase();

if(result) {
    console.log('The strings are similar.');
} else {
    console.log('The strings are not similar.');
}
```

Run Code

Output

```
The strings are similar.
```

In the above program, two strings are compared. Here,

- The `toUpperCase()` method converts all the string characters to uppercase.
- `===` is used to check if both the strings are the same.
- The `if...else` statement is used to display the result as per the condition.

> **Note**: You can also use the `toLowerCase()` method to convert all the strings to lowercase and perform the comparison.

Example 2: JS String Comparison Using RegEx

```
// program to perform string comparison

const string1 = 'JavaScript Program';
const string2 = 'javascript program';

// create regex
const pattern = new RegExp(string1, "gi");

// compare the stings
const result = pattern.test(string2)

if(result) {
    console.log('The strings are similar.');
} else {
    console.log('The strings are not similar.');
}
```

Run Code

Output

```
The strings are similar.
```

In the above program, the RegEx is used with the `test()` method to perform case insensitive string comparison.

In the RegEx pattern, "g" syntax denotes **global** and "gi" syntax denotes **case insensitive** comparisons.

Example 3: Using localeCompare()

```
// program to perform case insensitive string comparison

const string1 = 'JavaScript Program';
const string2 = 'javascript program';

const result = string1.localeCompare(string2, undefined, { sensitivity: 'base' });

if(result == 0) {
    console.log('The strings are similar.');
} else {
```

```
    console.log('The strings are not similar.');
}
```
Run Code

Output

```
The strings are similar.
```

In the above program, the `localeCompare()` method is used to perform case insensitive string comparison.

The `localeCompare()` method returns a number that indicates whether a reference string comes before, or after, or is the same as the given string.

Here, `{ sensitivity: 'base' }` treats **A** and **a** as the same.

JavaScript Program to Encode a String to Base64

Example 1: Encode a String to Base64 Using btoa()

```
// program to encode a string to Base64
// defining the string
const str = "Learning JavaScript";

// encoding the string
const result = window.btoa(str);
console.log(result);

// decoding the string
const result1 = window.atob(result);
console.log(result1);
```
Run Code

Output

```
TGVhcm5pbmcgSmF2YVNjcmlwdA==
Learning JavaScript
```

In the above example, the `btoa()` method is used to convert the string to **Base64**.

The `atob()` method is used to convert the **Base64** to a string.

Example 2: Encode a String to Base64 Using Base64 Object

```
// program to encode a string to Base64
// create Base64 Object
const Base64 = {
// private property
_keyStr : "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=",

// public method for encoding
encode : function (input) {
    let output = "";
    let chr1, chr2, chr3, enc1, enc2, enc3, enc4;
    let i = 0;

    input = Base64._utf8_encode(input);

    while (i < input.length) {

        chr1 = input.charCodeAt(i++);
        chr2 = input.charCodeAt(i++);
        chr3 = input.charCodeAt(i++);

        enc1 = chr1 >> 2;
        enc2 = ((chr1 & 3) << 4) | (chr2 >> 4);
        enc3 = ((chr2 & 15) << 2) | (chr3 >> 6);
        enc4 = chr3 & 63;

        if (isNaN(chr2)) {
            enc3 = enc4 = 64;
        } else if (isNaN(chr3)) {
            enc4 = 64;
        }

        output = output +
        Base64._keyStr.charAt(enc1) + Base64._keyStr.charAt(enc2) +
        Base64._keyStr.charAt(enc3) + Base64._keyStr.charAt(enc4);

    }

    return output;
},

// public method for decoding
decode : function (input) {
    let output = "";
    let chr1, chr2, chr3;
    let enc1, enc2, enc3, enc4;
    let i = 0;
```

```
    input = input.replace(/[^A-Za-z0-9\+\/\=]/g, "");

    while (i < input.length) {

        enc1 = Base64._keyStr.indexOf(input.charAt(i++));
        enc2 = Base64._keyStr.indexOf(input.charAt(i++));
        enc3 = Base64._keyStr.indexOf(input.charAt(i++));
        enc4 = Base64._keyStr.indexOf(input.charAt(i++));

        chr1 = (enc1 << 2) | (enc2 >> 4);
        chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
        chr3 = ((enc3 & 3) << 6) | enc4;

        output = output + String.fromCharCode(chr1);

        if (enc3 != 64) {
            output = output + String.fromCharCode(chr2);
        }
        if (enc4 != 64) {
            output = output + String.fromCharCode(chr3);
        }

    }

    output = Base64._utf8_decode(output);

    return output;

},

// private method for UTF-8 encoding
_utf8_encode : function (string) {
    string = string.replace(/\r\n/g,"\n");
    let utftext = "";

    for (let n = 0; n < string.length; n++) {

        let c = string.charCodeAt(n);

        if (c < 128) {
            utftext += String.fromCharCode(c);
        }
        else if((c > 127) && (c < 2048)) {
            utftext += String.fromCharCode((c >> 6) | 192);
            utftext += String.fromCharCode((c & 63) | 128);
        }
        else {
            utftext += String.fromCharCode((c >> 12) | 224);
            utftext += String.fromCharCode(((c >> 6) & 63) | 128);
            utftext += String.fromCharCode((c & 63) | 128);
```

```
        }

    }

    return utftext;
},

// private method for UTF-8 decoding
_utf8_decode : function (utftext) {
    let string = "";
    let i = 0;
    let c = c1 = c2 = 0;

    while ( i < utftext.length ) {

        c = utftext.charCodeAt(i);

        if (c < 128) {
            string += String.fromCharCode(c);
            i++;
        }
        else if((c > 191) && (c < 224)) {
            c2 = utftext.charCodeAt(i+1);
            string += String.fromCharCode(((c & 31) << 6) | (c2 & 63));
            i += 2;
        }
        else {
            c2 = utftext.charCodeAt(i+1);
            c3 = utftext.charCodeAt(i+2);
            string += String.fromCharCode(((c & 15) << 12) | ((c2 & 63) << 6) | (c3 & 63));
            i += 3;
        }

    }
    return string;
}
}

// define the string
const string = 'Learning JavaScript';

// encode the String
const encodedString = Base64.encode(string);
console.log(encodedString);

// decode the String
const decodedString = Base64.decode(encodedString);
console.log(decodedString);
```

Run Code

Output

```
TGVhcm5pbmcgSmF2YVNjcmlwdA==
Learning JavaScript.
```

The `encode()` method encodes a string to Base64. The `decode()` method decodes the Base64 to a string.

JavaScript Program to Replace All Line Breaks with

Example 1: Replace All Line Breaks Using RegEx

```
// program to replace all line breaks in a string with <br>
const string = `I am Learning JavaScript.
JavaScript is fun.
JavaScript is easy.`;

const result = string.replace(/(\r\n|\r|\n)/g, '<br>');

console.log(result);
```

Run Code

Output

```
I am Learning JavaScript.<br>JavaScript is fun.<br>JavaScript is easy.
```

In the above example:

- The RegEx is used with the `replace()` method to replace all the line breaks in `string` with **
**.
- The pattern `/(\r\n|\r|\n)/` checks for line breaks.
- The pattern `/g` checks across all the string occurrences.

Example 2: Replace All Line Breaks Using Built-in Methods

```
// program to replace all line breaks in a string with <br>
const string = `I am Learning JavaScript.
```

```
JavaScript is fun.
JavaScript is easy.`;

const result = string.split('\n').join('<br>');

console.log(result);
```
Run Code

Output

```
I am Learning JavaScript.<br>JavaScript is fun.<br>JavaScript is easy.
```

In the above example, the built-in methods are used to replace all line breaks with **
**.

The `split('\n')` splits the string into array elements by splitting on a line break.

```
["I am Learning JavaScript.", "JavaScript is fun.", "JavaScript is easy."]
```

The `join('<br>')` method joins the array by adding `<br>` between array elements.

```
I am Learning JavaScript.<br>JavaScript is fun.<br>JavaScript is easy.
```

JavaScript Program to Get File Extension

Example 1: Using split() and pop()

```
// program to get the file extension

function getFileExtension(filename){

    // get file extension
    const extension = filename.split('.').pop();
    return extension;
}

// passing the filename
const result1 = getFileExtension('module.js');
console.log(result1);

const result2 = getFileExtension('module.txt');
console.log(result2);
```
Run Code

Output

```
js
txt
```

In the above program, the extension of the filename is extracted using the `split()` method and the `pop()` method.

- The filename is split into individual array elements using the `split()` method.
  Here, `filename.split('.')` gives `["module", "js"]` by splitting the string.
- The last array element, which is the extension, is returned using the `pop()` method.

Example 2: Using substring() and lastIndexOf()

```
// program to get the file extension

function getFileExtension(filename){
    // get file extension
    const extension = filename.substring(filename.lastIndexOf('.') + 1, filename.length);
    return extension;
}

const result1 = getFileExtension('module.js');
console.log(result1);

const result2 = getFileExtension('test.txt');
console.log(result2);
```

Run Code

Output

```
js
txt
```

In the above program, the extension of the filename is extracted using the `substring()` method and the `lastIndexOf()` method.

- `filename.lastIndexOf('.') + 1` returns the last position of `.` in the filename.
  1 is added because the position count starts from **0**.
- The `filename.length` property returns the length of the string.
- `substring(filename.lastIndexOf('.') + 1, filename.length)` method returns characters between the given indexes. For example, `'module.js'.substring(8, 10)` returns `js`.

- The **OR** `||` operator is used to return the original string if there is no `.` in the filename.

JavaScript Program to Generate a Range of Numbers and Characters

Example: Generate Range of Characters

```
// program to generate range of numbers and characters
function* iterate(a, b) {
  for (let i = a; i <= b; i += 1) {
    yield i
  }
}

function range(a, b) {
    if(typeof a === 'string') {
        let result = [...iterate(a.charCodeAt(), b.charCodeAt())].map(n =>
String.fromCharCode(n));
        console.log(result);
    }
    else {
        let result = [...iterate(a, b)];
        console.log(result);
    }
}

range(1, 5);
range('A', 'G');
```

Run Code

Output

```
[1, 2, 3, 4, 5]
["A", "B", "C", "D", "E", "F", "G"]
```

In the above program, a range of numbers and characters is generated between the upper and the lower bounds.

- The `iterate` generator function is used to iterate through lower and upper bounds.

- The spread syntax ... is then used to include all the elements returned by the `iterate` function.
- The `charCodeAt()` method takes in an index value and returns an integer representing its UTF-16 (16-bit Unicode Transformation Format) code.
- The `map()` method iterates through all the array elements.
- The `fromCharCode()` method converts Unicode values into characters.

Whitespaces From a Text

Example 1: Using split() and join()

```
// program to trim a string

const string = '      Hello World       ';

const result = string.split(' ').join('');

console.log(result);
```

Run Code

Output

```
HelloWorld
```

In the above program,

- The `split(' ')` method is used to split the strings into individual array elements.

```
["", "", "", "", "", "", "Hello", "World", "", "", "", "", "", "", ""]
```

- The `join('')` method merges the array into a string.

Example 2: Using Regular Expression

```
// program to trim a string

function trimString(x) {

    const result = x.replace(/\s/g,'');
    return result

}

const result = trimString('    Hello World    ');
console.log(result);
```

Run Code

Output

```
HelloWorld
```

In the above program, the Regular Expression is used with the `replace()` method to remove all whitespaces from a text.

`/\s/g` checks for whitespace in the string.