Strings and its properties

Strings:

In JavaScript, the string is a primitive data type that represents a sequence of characters. It is used to store and manipulate textual data. Strings are immutable, which means they cannot be changed once they are created. However, you can perform various operations on strings to create new strings or extract information from existing strings.

```
var str = "Hello World";
str[0]="h";
console.log(str); // will be "Hello World"
```

In JavaScript, strings can be created using single quotes ('), double quotes ("), or backticks (`). Here are some examples of string literals:

```
let str1 = 'Hello';          // Using single quotes
let str2 = "World";          // Using double quotes
let str3 = `JavaScript`;     // Using backticks
```

Strings in JavaScript have many built-in properties and methods that allow you to manipulate and work with them. Some commonly used string methods include charAt, substring, toUpperCase, toLowerCase, split, concat, and replace, among others.

chatAt()

In JavaScript, the charAt() method is used to retrieve the character at a specific index within a string. It returns the character at the specified index as a new string.

Here's the syntax of the charAt() method:

```
string.charAt(index)
```

Here's an example that demonstrates the usage of charAt():

```
let str = 'Hello, World!';
console.log(str.charAt(0)); // Output: "H"
console.log(str.charAt(7)); // Output: "W"
console.log(str.charAt(13)); // Output: "d"
console.log(str.charAt(20)); // Output: ""
```

String search

The search() method searches a string for a specified value and returns the index of the first occurrence of the value, or -1 if the value is not found.

```
let str = 'Hello, World!';
let searchString = 'World';
let index = str.search(searchString);
console.log(index);  // Output: 7
```

String literals

In JavaScript, string literals are sequences of characters enclosed in single quotes (') or double quotes ("). String literals are used to represent text and can contain any combination of letters, numbers, symbols, and whitespace.

```
let str1 = 'Hello, World!';
let str2 = "JavaScript is awesome.";
let str3 = '12345';
let str4 = "Special characters: !@#$%^&*()";
let str5 = "Whitespace characters: \t\t\n";
```

JavaScript provides template literals (also known as template strings), which are enclosed in backticks (`). Template literals allow for embedded expressions using ${} that are evaluated and replaced with their corresponding values. This provides a more convenient way to create strings with dynamic content. For example:

```
const name = "John";
const age = 30;

const templateString = `My name is ${name} and I am
${age} years old.`;
// Result: "My name is John and I am 30 years old."
```

Slice() , substring() and substr()

In JavaScript, there are several methods available to extract parts of a string: slice(), substring(), and substr(). These methods are used to retrieve substrings from a given string based on specified start and end positions or lengths. Here's a brief explanation of each method:

1) slice(startIndex, endIndex):

Extracts a portion of a string and returns a new string.
The startIndex is inclusive, and the endIndex is exclusive (up to, but not including).
If the endIndex is omitted, the slice extends to the end of the string.
Negative values for startIndex and endIndex are treated as offsets from the end of the string.
Example usage:

```
let str = 'Hello, World!';
let sliced = str.slice(7, 12); // "World"
```

2) substring(startIndex, endIndex):

Extracts a portion of a string and returns a new string.
The startIndex is inclusive, and the endIndex is exclusive (up to, but not including).
If the endIndex is omitted, the substring extends to the end of the string.
Negative values for startIndex and endIndex are treated as 0.
Example usage:

```
let str = 'Hello, World!';
let sub = str.substring(7, 12); // "World"
```

3) substr(startIndex, length):

Extracts a specified number of characters from a string, starting at the specified index, and returns a new string.
The startIndex is the starting index for the extraction.
The length is the number of characters to extract.
If the length is omitted, substr extracts characters to the end of the string.
Negative values for startIndex are treated as offsets from the end of the string.
Example usage:

```
let str = 'Hello, World!';
let substr = str.substr(7, 5); // "World"
```

It's important to note that slice(), substring(), and substr() all return a new string without modifying the original string. These methods provide flexibility when working with strings and allow you to extract substrings based on different requirements.

StartsWith()

The startsWith() method is a built-in JavaScript string method that allows you to check if a string starts with a specified substring. It returns a boolean value indicating whether the string begins with the specified characters.

The syntax of the startsWith() method is as follows:

```
string.startsWith(searchString[, position])
```

searchString: This parameter specifies the substring to search for at the beginning of the string.
position (optional): This parameter denotes the position within the string at which to start the search. If omitted, the default value is 0.

The startsWith() method performs a case-sensitive comparison. It returns true if the string starts with the specified substring, and false otherwise.

Here are a few examples to illustrate the usage of startsWith():

```
let str = 'Hello, world!';

console.log(str.startsWith('Hello')); // true
console.log(str.startsWith('Hello,')); // true
console.log(str.startsWith('hello')); // false (case-
sensitive)
console.log(str.startsWith('world', 7)); // true
(start searching from position 7)
```

toUpperCase() and toLowerCase()

1) toUpperCase():

The toUpperCase() method is used to convert all characters in a string to uppercase.
It does not modify the original string, but instead returns a new string with all uppercase characters.
Syntax: string.toUpperCase()
Example:

```
let str = 'Hello, World!';
let upperCaseStr = str.toUpperCase();
console.log(upperCaseStr); // Output: "HELLO, WORLD!"
```

2) toLowerCase():

The toLowerCase() method is used to convert all characters in a string to lowercase.

Like toUpperCase(), it also returns a new string with all lowercase characters and does not modify the original string.

Syntax: string.toLowerCase()

Example:

```
let str = 'Hello, World!';
let lowerCaseStr = str.toLowerCase();
console.log(lowerCaseStr); // Output: "hello, world!"
```

replace()

In JavaScript, the replace() method is used to replace occurrences of a specified substring or pattern within a string with a new substring. It returns a new string with the replacements made, while the original string remains unchanged.

```
let str = 'Hello, World!';
let newStr = str.replace('Hello', 'Hi');
console.log(newStr); // Output: "Hi, World!"
```

we replace the substring "Hello" with "Hi" in the string str using the replace() method. The resulting string is stored in the newStr variable

concat()

In JavaScript, the concat() method is used to combine two or more strings and return a new concatenated string. This method does not modify the original strings but instead creates a new string that contains the concatenated values. It is particularly useful when you want to combine multiple strings without altering the originals.

Here's the syntax of the concat() method:

```
const newString = string1.concat(string2, string3,
..., stringN);
```

Example:

```
const firstName = "John";
const lastName = "Doe";

const fullName = firstName.concat(" ", lastName);
console.log(fullName); // Output: "John Doe"
```

trim()

In JavaScript, the trim() method is used to remove leading and trailing whitespace (spaces, tabs, and newline characters) from a string. It does not modify the original string but instead returns a new string with the whitespace removed.

Here's the syntax of the trim() method:

```
const trimmedString = originalString.trim();
```

Example:

```
const textWithWhitespace = "   Hello, world!   ";
const trimmedText = textWithWhitespace.trim();
console.log(trimmedText); // Output: "Hello, world!"
```

IndexOf()

In JavaScript, the indexOf() method is used to find the index of the first occurrence of a specified value within a string. If the value is found, the method returns the index (a non-negative integer). If the value is not found, the method returns -1.

Here's the syntax of the indexOf() method:

```
const index = originalString.indexOf(searchValue);
```

Example:

```
const sentence = "The quick brown fox jumps over the
lazy dog.";

const index1 = sentence.indexOf("quick");
console.log(index1); // Output: 4

const index2 = sentence.indexOf("cat");
console.log(index2); // Output: -1 (not found)
```