JavaScript String Methods

JavaScript String Length

Definition:

- **JavaScript uses the length property to specify the number of characters in a string. Returns the total number of characters, including spaces, in the specified string.**

- **The length property is a built-in property of the JavaScript String object.**

Usage:

- **You can access the length property of a string by appending .length to the string variable or string literal.**

Example:

String:

```
let str = "Hello, World!";

console.log(str.length); // Output: 13
```

Empty String:

let emptyString = "";

console.log(emptyString.length); // Output: 0

String with Spaces:

let stringWithSpaces = " Hello ";

console.log(stringWithSpaces.length); // Output: 11 (counting spaces as characters)

String with Special Characters:

let stringWithSpecialChars = "Hello, World! ♥";

console.log(stringWithSpecialChars.length); // Output: 15 (including the heart symbol)

String with Non-Latin Characters (Unicode):

let stringWithUnicode = "こんにちは"; // Japanese greeting "Konnichiwa"

console.log(stringWithUnicode.length); // Output: 5

String with Emoji Characters:

```
let stringWithEmoji = "😊🌟";

console.log(stringWithEmoji.length); // Output: 2 (even though it looks like one character, some emojis require multiple Unicode code points)
```

Special Considerations:

1. **It counts all characters including whitespace.**
2. **It is a property, not a method, so no parentheses () are needed to access it.**
3. **It returns the number of 16-bit Unicode characters in the string. Certain characters might occupy more than one position in the string due to being represented in Unicode with more than one code unit (such as emoji characters).**
4. **It's a property, so you don't invoke it like a function.**
5. **Immutable: The length property does not change the original string. It's a read-only property that simply reflects the current length of the string.**

JavaScript String charAt()

Definition:

- **In JavaScript, the charAt() method is used to retrieve a character at the index specified in a string. The index is given as a parameter to the charAt() method, and it returns the characters at that index position.**
- **charAt() is a method of the JavaScript String object.**

Usage:

You call charAt() on a string by providing the index of the character you want to retrieve.

Syntax:

- **the syntax for charAt() is: string.charAt(index).**
- **string: The string from which you want to retrieve a character.**
- **index: The zero-based index of the character you want to retrieve.**

Return value:

It returns the character located at the specified index position within the string. If the index is out of range (less than 0 or greater than or equal to the length of the string), an empty string is returned.

Special considerations:

- **If the index is negative or greater than or equal to the length of the string, charAt() returns an empty string.**
- **The index parameter is zero-based, meaning the first character in the string is at index 0, the second character is at index 1, and so on.**
- **If the index is not an integer, it will be rounded down before accessing the character.**
- **If you try to access characters outside the string's length, it returns an empty string.**
- **charAt() is useful for scenarios where you need to access individual characters within a string for manipulation or analysis.**

Example:

Accessing Characters in a String:

```
let str = "Hello, World!";

console.log(str.charAt(0)); // Output: "H"

console.log(str.charAt(7)); // Output: "W"

console.log(str.charAt(12)); // Output: "!"
```

Using Negative Index:

```
let str = "Hello, World!";

console.log(str.charAt(-1)); // Output: ""
```

Using Decimal Index:

```
let str = "Hello, World!";

console.log(str.charAt(4.7)); // Output: "o"
```

Accessing Characters Beyond String Length:

```
let str = "Hello, World!";

console.log(str.charAt(20)); // Output: ""
```

Looping Through Characters:

```
let str = "Hello";

for (let i = 0; i < str.length; i++) {

   console.log(str.charAt(i)); // Output: Each character in "Hello" on separate lines

}
```

Using charAt() with Variable Index:

```
let str = "Hello";

let index = 2;

console.log(str.charAt(index)); // Output: "l"
```

JavaScript String charCodeAt()

Definition:

- **The charCodeAt() method in JavaScript returns the Unicode value of the character at a specified index within a string. This method is useful for getting the Unicode value of a character at a particular position in a string.**
- **charCodeAt() is a method of the JavaScript String object.**

Usage:

- **You call charCodeAt() on a string by providing the index of the character whose Unicode value you want to retrieve.**

Syntax:

- **The syntax for charCodeAt() is: string.charCodeAt(index).**
- **string: The string from which you want to retrieve the Unicode value of a character.**
- **index: The zero-based index of the character whose Unicode value you want to retrieve.**

Return value:

It returns the Unicode value of the character located at the specified index position within the string. If the index is out of range (less than 0 or greater than or equal to the length of the string), NaN (Not-a-Number) is returned.

Special considerations:

- **The index parameter is zero-based, meaning the first character in the string is at index 0, the second character is at index 1, and so on.**
- **If the index is negative or greater than or equal to the length of the string, NaN is returned.**
- **If the character at the specified index consists of more than one 16-bit unit in UTF-16 encoding (e.g., for characters with code points greater than 0xFFFF), charCodeAt() returns the UTF-16 surrogate pair code of the character.**
- **If you need the actual character itself rather than its Unicode value, you can use charAt() method instead.**
- **charCodeAt() is particularly useful when dealing with string manipulation tasks that require working with Unicode values of individual characters within a string.**

Example:

Accessing Unicode in a String:

```
let str = "Hello";

console.log(str.charCodeAt(0)); // Output: 72 (Unicode value of "H")

console.log(str.charCodeAt(1)); // Output: 101 (Unicode value of "e")
```

Using Negative Index:

```
let str = "Hello";

console.log(str.charCodeAt(-1)); // Output: NaN
```

Using Decimal Index:

```
let str = "Hello";

console.log(str.charCodeAt(2.5)); // Output: 108 (Unicode value of "l")
```

Accessing Unicode Beyond String Length:

```
let str = "Hello";

console.log(str.charCodeAt(20)); // Output: NaN
```

Looping Through Characters and Getting Unicode Values:

```
let str = "Hello";

for (let i = 0; i < str.length; i++) {

   console.log(str.charCodeAt(i)); // Output: Unicode value of each character in "Hello"

}
```

Using charCodeAt() with Variable Index:

```
let str = "Hello";

let index = 2;

console.log(str.charCodeAt(index)); // Output: 108 (Unicode value of "l")
```

JavaScript String at()

- **ES2022 introduced the string method at()**
- **The at() method returns the character at a specified index (position) in a string.**
- **The at() method is supported in all modern browsers since March 2022.**

Note:

The at() method is a new addition to JavaScript.

It allows the use of negative indexes while charAt() do not.

Now you can use myString.at(-2) instead of charAt(myString.length-2).

Example:

```
const name = "Hello";
let letter = name.at(2);
console.log(letter); // l
console.log(name.at(-4)); // e
console.log(name.length-1) // 4
```

JavaScript String slice()

Definition:

- **The slice() method in JavaScript is used to extract a portion of a string and return it as a new string, without modifying the original string. Allows you to extract substrings based on specified start and end indices. slice() is a method of the JavaScript String object.**

Usage:

- **You call slice() on the string and specify the start and end index positions to extract the substring.**

Syntax:

- **The syntax for slice() is: string.slice(startIndex, endIndex).**
- **string: The original string from which you want to extract a substring.**
- **startIndex: The zero-based index at which to begin extraction. If negative, it counts from the end of the string.**
- **endIndex: Optional. The zero-based index before which to end extraction. slice() extracts up to but not including endIndex. If omitted, slice() extracts to the end of the string.**

Return value:

It returns a new string containing the extracted phase of the original string.

Special considerations:

- **If startIndex is greater than or equal to endIndex, slice() returns an empty string.**
- **If startIndex or endIndex is negative, it counts from the end of the string. -1 refers to the last character, -2 refers to the second last character, and so on.**
- **If startIndex is greater than the length of the string, slice() returns an empty string. If endIndex is greater than the length of the string, slice() extracts characters up to the end of the string.**
- **slice() is commonly used for extracting substrings based on specified indices, such as when parsing strings or manipulating text data.**

Example:

Extracting Substring from the Beginning:

```
let str = "Hello, World!";

let slicedStr = str.slice(0, 5);

console.log(slicedStr); // Output: "Hello"
```

Extracting Substring from the End:

```
let str = "Hello, World!";

let slicedStr = str.slice(-6);

console.log(slicedStr); // Output: "World!"
```

Omitting the End Index (Extracting till the End of String):

```
let str2 = "Hello, World!";

let slicedStr2 = str.slice(7);

console.log(slicedStr2); // Output: "World!"
```

Negative Start and End Index:

```
let str3 = "Hello, World!";

let slicedStr3 = str3.slice(-6, -1);

console.log(slicedStr3); // Output: "World"

console.log(str3.slice(-6,-2)) // Output: "Worl"
```

Start Index Greater than End Index (Returns an Empty String):

```
let str = "Hello, World!";

let slicedStr = str.slice(7, 5);

console.log(slicedStr); // Output: ""
```

Extracting Last Character:

```
let str = "Hello, World!";

let slicedStr = str.slice(-1);

console.log(slicedStr); // Output: "!"
```

If you omit(removed or leave out) the second parameter, the method will slice out the rest of the string:

```
let fruits = "Apple, Banana, Kiwi";

console.log(fruits.slice(7));
```

If a parameter is negative, the position is counted from the end of the string:

```
let fruits = "Apple, Banana, Kiwi";

console.log(fruits.slice(-12));
```

This example slices out a portion of a string from position -12 to position -6:

```
let fruits = "Apple, Banana, Kiwi";

console.log(fruits.slice(-12,-6));
```

JavaScript String substr()

Definition:

- **In JavaScript, the substr() method is used to extract a portion of a string, starting from a specified index position, and with an optional length. This method returns the extracted substring as a new string without modifying the original string.**
- **substr() is a method of the JavaScript String object.**
- **substring() is similar to slice().**
- **The difference is that start and end values less than 0 are treated as 0 in substring().**

Usage:

- **You call substr() on a string and specify the starting index position from which you want to extract the substring, along with an optional length of characters to extract.**

Syntax:

- **The syntax for substr() is: string.substr(startIndex, length).**
- **string: The original string from which you want to extract a substring.**
- **startIndex: The zero-based index at which to begin extraction. If negative, it counts from the end of the string.**
- **length: Optional. The number of characters to extract. If omitted, substr() extracts characters to the end of the string.**

Return value:

It returns a new string containing the extracted substring.

Special considerations:

- **If startIndex is negative, it counts from the end of the string. -1 refers to the last character, -2 refers to the second last character, and so on.**
- **If startIndex is greater than or equal to the length of the string, substr() returns an empty string.**
- **If length is specified, the substring will have a maximum length of length characters starting from the startIndex.**
- **If length is negative or 0, an empty string is returned.**
- **substr() is commonly used for extracting substrings from strings based on a starting index position and an optional length, providing flexibility in manipulating and extracting portions of text data.**

Example:

Extracting Substring from the Beginning:

```
let str = "Hello, World!";

let sub = str.substr(0, 5);

console.log(sub); // Output: "Hello"
```

Extracting Substring from the End using Negative Index:

```
let str = "Hello, World!";

let sub = str.substr(-6);

console.log(sub); // Output: "World!"
```

Omitting the Length Parameter (Extracting till the End of String):

```
let str = "Hello, World!";

let sub = str.substr(7);

console.log(sub); // Output: "World!"
```

Negative Start Index (Counting from the End of String):

```
let str = "Hello, World!";

let sub = str.substr(-6, 4);

console.log(sub); // Output: "Worl"
```

Using Length Parameter to Limit the Extracted Substring:

```
let str = "Hello, World!";

let sub = str.substr(7, 5);

console.log(sub); // Output: "World"
```

Omitting Length Parameter (Extracting till the End of String from a Negative Start Index):

```
let str = "Hello, World!";

let sub = str.substr(-6);

console.log(sub); // Output: "World!"
```

If you omit the second parameter, substr() will slice out the rest of the string.

```
let str = "Apple, Banana, Kiwi";

let part = str.substr(7);
```

If the first parameter is negative, the position counts from the end of the string.

```
let str = "Apple, Banana, Kiwi";

let part = str.substr(-4);
```

Converting to Upper and Lower Case

A string is converted to upper case with `toUpperCase()`:

Example:

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase();
```

A string is converted to lower case with `toLowerCase()`:

Example:

```
let text1 = "Hello World!";       // String
let text2 = text1.toLowerCase();  // text2 is text1
converted to lower
```

JavaScript String concat()

`concat()` joins two or more strings:

Example:

```
let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2);
```

The `concat()` method can be used instead of the plus operator. These two lines do the same:

Example:

```
text = "Hello" + " " + "World!";
text = "Hello".concat(" ", "World!");
```

Note

All string methods return a new string. They don't modify the original string.

Formally said:

Strings are immutable: Strings cannot be changed, only replaced.

JavaScript String trim()

The `trim()` method removes whitespace from both sides of a string:

Example:

```
let text1 = "      Hello World!      ";
let text2 = text1.trim();
```

JavaScript String trimStart()

The `trimStart()` method works like `trim()`, but removes whitespace only from the start of a string.

Example:

```
let text1 = "     Hello World!     ";
let text2 = text1.trimStart();
```

JavaScript String trimEnd()

The `trimEnd()` method works like `trim()`, but removes whitespace only from the end of a string.

Example:

```
let text1 = "     Hello World!     ";
let text2 = text1.trimEnd()
```

JavaScript String padStart()

- **The `padStart()` method pads a string from the start.**
- **It pads a string with another string (multiple times) until it reaches a given length.**
- **In JavaScript, the padStart() method is used to pad the current string with another string (repeated as necessary) until the resulting string reaches the specified length. The padding is applied from the start (left) of the original string.**

Syntax:

- **The syntax for padStart() is: string.padStart(targetLength, padString).**
- **string: The original string to pad.**
- **targetLength: The length of the resulting padded string. If the original string is already longer than targetLength, no padding is applied.**
- **padString: Optional. The string to pad the original string with. If omitted, the space character (" ") is used.**

Return value:

It returns a new string that is the original string padded with the specified pad string until it reaches the desired length

Example:

```
let str = "Hello";

let paddedStr = str.padStart(10, " ");

console.log(paddedStr); // Output: "     Hello"
```

Example:

Pad a string with "0" until it reaches the length 4:

```
let text = "5";
let padded = text.padStart(4,"0");
```

Example:

Pad a string with "x" until it reaches the length 4:

```
let text = "5";
let padded = text.padStart(4,"x");
```

JavaScript String padEnd()

The `padEnd()` method pads a string from the end.

It pads a string with another string (multiple times) until it reaches a given length.

Example:

```
let text = "5";
let padded = text.padEnd(4,"0");
```

JavaScript String repeat()

The `repeat()` method returns a string with a number of copies of a string.

The `repeat()` method returns a new string.

The `repeat()` method does not change the original string.

Syntax

```
string.repeat(count)
```

Parameters

Parameter	Description
count	Required. The number of copies wanted.

Return Value

Type	Description
String	A new string containing the copies.

Example:

Create copies of a text:

```
let text = "Hello world!";
let result = text.repeat(2);
```

Example:

Create copies of a text:

```
let text = "Hello world!";
let result = text.repeat(4);
```

Replacing String Content

The `replace()` method replaces a specified value with another value in a string:

```
let text = "Please visit Microsoft!";
let newText = text.replace("Microsoft", "W3Schools");
```

Note

The `replace()` method does not change the string it is called on.

The `replace()` method returns a new string.

The `replace()` method replaces only the first **match**

If you want to replace all matches, use a regular expression with the /g flag set. See examples below.

By default, the `replace()` method replaces **only the first** match:

Example

```
let text = "Please visit Microsoft and Microsoft!";
let newText = text.replace("Microsoft", "W3Schools");
```

By default, the `replace()` method is case sensitive. Writing MICROSOFT (with upper-case) will not work:

Example

```
let text = "Please visit Microsoft!";
let newText = text.replace("MICROSOFT", "W3Schools");
```

To replace case insensitive, use a regular expression **with an `/i` flag (insensitive):**

Example

```
let text = "Please visit Microsoft!";
let newText = text.replace(/MICROSOFT/i, "W3Schools");
```

To replace all matches, use a regular expression **with a `/g` flag (global match):**

Example

```
let text = "Please visit Microsoft and Microsoft!";
let newText = text.replace(/Microsoft/g, "W3Schools");
```

JavaScript String ReplaceAll()

In 2021, JavaScript introduced the string method `replaceAll()`:

Example

```
let text = "I love cats. Cats are very easy to love. Cats are very popular."

text = text.replaceAll("Cats","Dogs");

text = text.replaceAll("cats","dogs");
```

The `replaceAll()` method allows you to specify a regular expression instead of a string to be replaced.

If the parameter is a regular expression, the global flag (g) must be set, otherwise a TypeError is thrown.

Example

```
let text = "I love cats. Cats are very easy to love. Cats are very popular."

text = text.replaceAll(/Cats/g,"Dogs");
text = text.replaceAll(/cats/g,"dogs");
```

Converting a String to an Array

If you want to work with a string as an array, you can convert it to an array.

JavaScript String split()

A string can be converted to an array with the `split()` method:

Example

```
text.split(",")    // Split on commas
text.split(" ")    // Split on spaces
text.split("|")    // Split on pipe
```

Example

```
let str6 = "a,b,c,d,e,f";
let parts = str6.split(",");
console.log(str6); //output a,b,c,d,e,f
console.log(parts); //output [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’]
console.log(str6[1]); // Output: ["Hello", "World"]
console.log(parts[3]);
```

JavaScript String Search

JavaScript String indexOf()

The `indexOf()` method returns the index (position) of the first occurrence of a string in a string, or it returns -1 if the string is not found:

Example

```
let text = "Please locate where 'locate' occurs!";
let index = text.indexOf("locate");
```

o/p - 7

Example

```
let text = "Please locate where 'locate' occurs!";
let index = text.indexOf("locateee");
```

o/p - -1

JavaScript String lastIndexOf()

The `lastIndexOf()` method returns the index of the last **occurrence of a specified text in a string:**

Example

```
let text = "Please locate where 'locate' occurs!";
let index = text.lastIndexOf("locate");
```

o/p - 21

Both methods accept a second parameter as the starting position for the search:

Example

```
let text = "Please locate where 'locate' occurs!";
let index = text.indexOf("locate", 15);
```

The `lastIndexOf()` methods searches backwards (from the end to the beginning), meaning: if the second parameter is `15`, the search starts at position 15, and searches to the beginning of the string.

Example

```
let text = "Please locate where 'locate' occurs!";
text.lastIndexOf("locate", 15);
```

JavaScript String search()

The `search()` method searches a string for a string (or a regular expression) and returns the position of the match:

Examples

```
let text = "Please locate where 'locate' occurs!";
text.search("locate");
```

Examples

```
let text = "Please locate where 'locate' occurs!";
text.search(/locate/);
```

Did You Notice?

The two methods, `indexOf()` and `search()`, are **equal?**

They accept the same arguments (parameters), and return the same value?

The two methods are **NOT** equal. These are the differences:

- The `search()` method cannot take a second start position argument.
- The `indexOf()` method cannot take powerful search values (regular expressions).

JavaScript String match()

The `match()` method returns an array containing the results of matching a string against a string (or a regular expression).

Examples

Perform a search for "ain":

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match("ain");
```

Perform a search for "ain":

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/);
```

Perform a global search for "ain":

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/g);
```

Perform a global, case-insensitive search for "ain":

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/gi);
```

JavaScript String matchAll()

The `matchAll()` method returns an iterator containing the results of matching a string against a string (or a regular expression).

```
// Example 1
let text4 = "I love cats. Cats are very easy to
love. Cats are very popular.";
const iterator = text4.matchAll("Cats");
```

```
// document.getElementById("demo").innerHTML =
Array.from(iterator);
console.log(Array.from(iterator));
```

If the parameter is a regular expression, the global flag (g) must be set, otherwise a TypeError is thrown.

```
// Example 2
console.log(Array.from(text4.matchAll(/Cats/g)));
```

If you want to search case insensitive, the insensitive flag (i) must be set:

```
// Example 3
console.log(Array.from(text4.matchAll(/Cats/gi)));
```

JavaScript String includes()

The `includes()` method returns true if a string contains a specified value.

Otherwise it returns `false`.

Examples

Check if a string includes "world":

```
let sen = "Hello world, welcome to the universe.";
console.log(sen.includes("world")); //true
```

Check if a string includes "world". Start at position 12:

```
console.log(sen.includes("world", 12)); //false
```

JavaScript String startsWith()

The `startsWith()` method returns `true` if a string begins with a specified value.

Otherwise it returns `false`:

```
// Examples
// Returns true:

let text6 = "Hello world, welcome to the universe.";
console.log(text6.startsWith("Hello"));
```

```
// Returns false:
console.log(text6.startsWith("world"));
```

A start position for the search can be specified:

```
//Returns false:
console.log(text6.startsWith("world", 5));
```

```
//Returns true:
console.log(text6.startsWith("world", 6));
```

JavaScript String endsWith()

The `endsWith()` method returns `true` if a string ends with a specified value.

Otherwise it returns `false`:

Examples

Check if a string ends with "Doe":

```
let name = "John Doe";
console.log(name.endsWith("Doe"));
```

Check if the 11 first characters of a string ends with "world":

```
let text = "Hello world, welcome to the universe.";
text.endsWith("world", 11);
```

JavaScript Template Strings

Back-Tics Syntax

Template Strings use back-ticks (``) rather than the quotes ("") to define a string:

Example

```
let text = `Hello World!`;
```

Quotes Inside Strings

Template Strings allow both single and double quotes inside a string:

Example

```
let text = `He's often called "Johnny"`;
```

Multiline Strings

Template Strings allow multiline strings:

Example

```
let text =
`The quick
brown fox
jumps over
the lazy dog`;
```

Interpolation

Template String provide an easy way to interpolate variables and expressions into strings.

The method is called string interpolation.

The syntax is:

```
${...}
```