JavaScript Strings

JavaScript Strings are useful for holding data that can be represented in text form.

A JavaScript string is zero or more characters written inside quotes.

Example:

```
 let text= “Keshav Memorial Institute of Technology.”
```

text→ variable name

Keshav Memorial Institute of Technology. → String literal encoded with double quotes

You can use single or double quotes:

```
let carName1= "BMWXC99";  //Doublequotes
let carName2= 'BMWXC98';  // Single quotes
```

String primitives and String objects:

Note that JavaScript distinguishes between String objects and **primitive string** values. (The same is true of **Boolean** and **Numbers**.)

String literals (denoted by double or single quotes) and strings returned from String calls in a non-constructor context (that is, called without using the **new** keyword) are primitive strings. In contexts where a method is to be invoked on a primitive string or a property lookup occurs, JavaScript will automatically wrap the string primitive and call the method or perform the property lookup on the wrapper object instead.

```
conststr = "foo"; // A literal is a string primitive
conststr2 = String(1); // Coerced into the string primitive "1"
conststr3 = String(true); // Coerced into the string primitive "true"
conststrObj = new String(str); // String with new returns a string wrapper
object.

console.log(typeofstr); // "string"
console.log(typeofstr2); // "string"
console.log(typeofstr3); // "string"
console.log(typeofstrObj); // "object"
```

String primitives and String objects also give different results when using **eval()**. Primitives passed to eval are treated as source code; String objects are treated as all other objects are, by returning the object. For example:

```
const s1 = "2 + 2"; // creates a string primitive
const s2 = new String("2 + 2"); // creates a String object
console.log(eval(s1)); // returns the number 4
console.log(eval(s2)); // returns the string "2 + 2"
                note: use node to run this
```

Strings are immutable:

It’s important to know that in JavaScript, strings are immutable. This means that once a string is created, its contents cannot be changed.

Instead, you must create a new string representing the modified version when you want to modify a string.

For example, if you have a string assigned to a variable, you cannot modify it. Instead, you will create a new string and assign the new string to the same variable like this:

```
let name ="John Doe";
name ="Jane";
```

This means that the original string "John Doe" still exists in memory, but the variable name now refers to the new string "Jane".

JavaScript Template Literals:

Back-Tics Syntax
Template Literals use back-ticks (``) rather than the quotes ("") to define a string:

Example:

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

Quotes Inside Strings

With **template literals**, you can use both single and double quotes inside a string:

Example

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

Multiline Strings

Template literals allows multiline strings:

Example

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

Expression Substitution

Template literals allow expressions in strings:

Example

```
let price= 10;
let VAT= 0.25;
let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`;     //Total: 12.50
```

JavaScript String Methods

String length	**String trim()**
String slice()	**String trimStart()**
String substring()	**String trimEnd()**
String substr()	**String padStart()**
String replace()	**String padEnd()**
String replaceAll()	**String charAt()**
String toUpperCase()	**String charCodeAt()**
String toLowerCase()	**String split()**
String concat()	

String Length

The **length** property returns the length of a string:

Example

```
let text= "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = text.length;
console.log(length);   // 26
```

Extracting String Parts
There are 3 methods for extracting a part of a string:
slice(start, end)
substring(start, end)
substr(start, length)

String slice()

slice() extracts a part of a string and returns the extracted part in a new string.

The method takes 2 parameters: start position, and end position (end not included).

Example

Slice out a portion of a string from position 7 to position 13:

```
let text= "Apple,Banana,Kiwi";
let part = text.slice(7, 13);
```

If you omit the second parameter, the method will slice out the rest of the string:

```
let text= "Apple,Banana,Kiwi";
let part = text.slice(7);
```

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

```
let text= "Apple,Banana,Kiwi";
let part = text.slice(-12);
```

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

```
let text= "Apple,Banana,Kiwi";
let part = text.slice(-12, -6);
```

String substring()

substring() is similar to **slice().**

The difference is that start and end values less than 0 are treated as 0 in **substring()**.

Example:

```
let str= "Apple,Banana,Kiwi";
let part = str.substring(7, 13);
```

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

Replacing String Content
The replace() method replaces a specified value with another value in a string:

Example

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

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.

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

Example

```
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.

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

String toUpperCase()

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase();   // text2 is text1 converted to Upper
HELLO WORLD!
```

String toLowerCase()

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

String concat()

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

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

```
text = "Hello" + " " + "World!";
text = "Hello".concat(" ", "World!");// both will give same result  Hello
World!
```

String split()

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

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

If the separator is omitted, the returned array will contain the whole string in index [0].
If the separator is "", the returned array will be an array of single characters:

```
text.split("")
```

String charCodeAt()

The charCodeAt() method returns the unicode of the character at a specified index in a string:
The method returns a UTF-16 code (an integer between 0 and 65535).

```
let text = "HELLO WORLD";
let char = text.charCodeAt(0);
```

Property Access

ECMAScript 5 (2009) allows property access [ ] on strings:

Example

```
let text = "HELLO WORLD";
let char = text[0];
```

Note
Property access might be a little unpredictable:
It makes strings look like arrays (but they are not)
If no character is found, [ ] returns undefined, while charAt() returns an empty string.
It is read only. str[0] = "A" gives no error (but does not work!)
Example
let text = "HELLO WORLD";
text[0] = "A"; // Gives no error, but does not work

Search Methods

- String indexOf()
- String lastIndexOf()
- String search()
- String match()
- String matchAll()
- String includes()
- String startsWith()
- String endsWith()

String indexOf()

The indexOf() method returns the index (position) the first occurrence of a string in a string:

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

Note

JavaScript counts positions from zero.

0 is the first position in a string, 1 is the second, 2 is the third, ..

String search()

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

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

JavaScript String match()

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

Perform a search for "ain":

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match("ain");  // array size =1
```

Perform a search for "ain":

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/);  //array size =1
```

Perform a global search for "ain":

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/g);   //array size =3
```

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

```
let text = "The rain in SPAIN stays mainly in the plain";
text.match(/ain/gi); // array size =4
```