STRING

The **String** object is used to represent and manipulate a sequence of characters.

Description

Strings are useful for holding data that can be represented in text form. Some of the most-used operations on strings are to check their length, to build and concatenate them using the + and += string operators, checking for the existence or location of substrings with the indexOf() method, or extracting substrings with the substring() method.

Creating strings

Strings can be created as primitives, from string literals, or as objects, using the String() constructor:

```
const string1 = "A string primitive";
const string2 = 'Also a string primitive';
const string3 = `Yet another string primitive`;
const string4 = new String("A String object");
```

String primitives and string objects share many behaviors, but have other important differences and caveats. See "String primitives and String objects" below.

String literals can be specified using single or double quotes, which are treated identically, or using the backtick character `. This last form specifies a template literal: with this form you can interpolate expressions. For more information on the syntax of string literals, see lexical grammar.

Comparing strings

Use the less-than and greater-than operators to compare strings.

```
const a = "a";
const b = "b";
if (a < b) {
  // true
  console.log(`${a} is less than ${b}`);
} else if (a > b) {
  console.log(`${a} is greater than ${b}`);
} else {
  console.log(`${a} and ${b} are equal.`);
}
```

Note that all comparison operators, including === and ==, compare strings case-sensitively. A common way to compare

strings case-insensitively is to convert both to the same case (upper or lower) before comparing them.

```
function areEqualCaseInsensitive(str1, str2) {
  return str1.toUpperCase() === str2.toUpperCase();
}
```

The choice of whether to transform by toUpperCase() or toLowerCase() is mostly arbitrary, and neither one is fully robust when extending beyond the Latin alphabet. For example, the German lowercase letter ß and ss are both transformed to SS by toUpperCase(), while the Turkish letter ı would be falsely reported as unequal to I by toLowerCase() unless specifically using toLocaleLowerCase("tr").

JSCopy to Clipboard

```
const areEqualInUpperCase = (str1, str2) =>
  str1.toUpperCase() === str2.toUpperCase();
const areEqualInLowerCase = (str1, str2) =>
  str1.toLowerCase() === str2.toLowerCase();

areEqualInUpperCase("ß", "ss"); // true; should be false
areEqualInLowerCase("ı", "I"); // false; should be true
```

A locale-aware and robust solution for testing case-insensitive equality is to use the Intl.Collator API or the string's localeCompare() method — they share the same

interface — with the sensitivity option set to "accent" or "base".

```
const areEqual = (str1, str2, locale = "en-US") =>
  str1.localeCompare(str2, locale, { sensitivity: "accent" }) ===
0;


areEqual("ß", "ss", "de"); // false
areEqual("ı", "I", "tr"); // true
```

The localeCompare() method enables string comparison in a similar fashion as strcmp() — it allows sorting strings in a locale-aware manner.

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.

```
const strPrim = "foo"; // A literal is a string primitive
const strPrim2 = String(1); // Coerced into the string primitive "1"
const strPrim3 = String(true); // Coerced into the string primitive "true"
const strObj = new String(strPrim); // String with new returns a string wrapper object.

console.log(typeof strPrim); // "string"
console.log(typeof strPrim2); // "string"
console.log(typeof strPrim3); // "string"
console.log(typeof strObj); // "object"
```

Warning:

You should rarely find yourself using String as a constructor.

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"
```

For these reasons, the code may break when it encounters String objects when it expects a primitive string instead, although generally, authors need not worry about the distinction.

A String object can always be converted to its primitive counterpart with the valueOf() method.

console.log(eval(s2.valueOf())); // returns the number 4

String coercion

Many built-in operations that expect strings first coerce their arguments to strings (which is largely why String objects behave similarly to string primitives). The operation can be summarized as follows:

- Strings are returned as-is.
- undefined turns into "undefined".
- null turns into "null".
- true turns into "true"; false turns into "false".
- Numbers are converted with the same algorithm as toString(10).
- BigInts are converted with the same algorithm as toString(10).
- Symbols throw a TypeError.

- Objects are first converted to a primitive by calling its [@@toPrimitive]() (with "string" as hint), toString(), and valueOf() methods, in that order. The resulting primitive is then converted to a string.

There are several ways to achieve nearly the same effect in JavaScript.

- Template literal: `${x}` does exactly the string coercion steps explained above for the embedded expression.
- The String() function: String(x) uses the same algorithm to convert x, except that Symbols don't throw a TypeError, but return "Symbol(description)", where description is the description of the Symbol.
- Using the + operator: "" + x coerces its operand to a *primitive* instead of a *string*, and, for some objects, has entirely different behaviors from normal string coercion. See its reference page for more details.
-

String Properties

Here is a list of the properties of String object and their description.

Sr.No.	Property & Description
1	constructor

	Returns a reference to the String function that created the object.
2	length Returns the length of the string.
3	prototype The prototype property allows you to add properties and methods to an object.

JavaScript String - constructor Property

Description

A constructor returns a reference to the string function that created the instance's prototype.

Syntax

Its syntax is as follows −

```
string.constructor
```

Return Value

Returns the function that created this object's instance.

Example

```
<html>
  <head>
    <title>JavaScript String constructor Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = new String( "This is string" );
      document.write("str.constructor is:" + str.constructor);
    </script>
  </body>
</html>
```

Output

```
str.constructor is: function String() { [native code] }
```

JavaScript String - length Property

Description

This property returns the number of characters in a string.

Syntax

Use the following syntax to find the length of a string −

```
string.length
```

Return Value

Returns the number of characters in the string.

Example

```
<html>
  <head>
    <title>JavaScript String length Property</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = new String( "This is string" );
      document.write("str.length is:" + str.length);
    </script>
  </body>
</html>
```

Output

```
str.length is:14
```

JavaScript Object - prototype

Description

The prototype property allows you to add properties and methods to any object (Number, Boolean, String and Date etc.).

Note – Prototype is a global property which is available with almost all the objects.

Syntax

Its syntax is as follows –

```
object.prototype.name = value
```

Example

```
<html>
  <head>
    <title>User-defined objects</title>
    <script type = "text/javascript">
      function book(title, author) {
        this.title = title;
        this.author  = author;
      }
    </script>
  </head>
  <body>
    <script type = "text/javascript">
```

```
      var myBook = new book("Perl", "Mohtashim");
      book.prototype.price = null;
      myBook.price = 100;

      document.write("Book title is : " + myBook.title +
"<br>");
      document.write("Book author is : " + myBook.author +
"<br>");
      document.write("Book price is : " + myBook.price +
"<br>");
    </script>
  </body>
</html>
```

Output

```
Book title is : Perl
Book author is : Mohtashim
Book price is : 100
```

JavaScript String - charAt() Method

Description

charAt() is a method that returns the character from the specified index.

Characters in a string are indexed from left to right. The index of the first character is 0, and the index of the last character in a string, called **stringName,** is stringName.length – 1.

Syntax

Use the following syntax to find the character at a particular index.

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

Argument Details

index – An integer between 0 and 1 less than the length of the string.

Return Value

Returns the character from the specified index.

Example

```
<html>
  <head>
    <title>JavaScript String charAt() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = new String( "This is string" );
      document.writeln("str.charAt(0) is:" + str.charAt(0));
```

```
        document.writeln("<br />str.charAt(1) is:" +
str.charAt(1));
        document.writeln("<br />str.charAt(2) is:" +
str.charAt(2));
        document.writeln("<br />str.charAt(3) is:" +
str.charAt(3));
        document.writeln("<br />str.charAt(4) is:" +
str.charAt(4));
        document.writeln("<br />str.charAt(5) is:" +
str.charAt(5));
    </script>
  </body>
</html>
```

Output

```
str.charAt(0) is:T
str.charAt(1) is:h
str.charAt(2) is:i
str.charAt(3) is:s
str.charAt(4) is:
str.charAt(5) is:i
```

JavaScript String - charCodeAt() Method

Description

This method returns a number indicating the Unicode value of the character at the given index.

Unicode code points range from 0 to 1,114,111. The first 128 Unicode code points are a direct match of the ASCII character encoding. **charCodeAt()** always returns a value that is less than 65,536.

Syntax

Use the following syntax to find the character code at a particular index.

```
string.charCodeAt(index);
```

Argument Details

index − An integer between 0 and 1 less than the length of the string; if unspecified, defaults to 0.

Return Value

Returns a number indicating the Unicode value of the character at the given index. It returns NaN if the given index is not between 0 and 1 less than the length of the string.

Example

```
<html>
  <head>
    <title>JavaScript String charCodeAt() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = new String( "This is string" );
      document.write("str.charCodeAt(0) is:" +
str.charCodeAt(0));
      document.write("<br />str.charCodeAt(1) is:" +
str.charCodeAt(1));
      document.write("<br />str.charCodeAt(2) is:" +
str.charCodeAt(2));
      document.write("<br />str.charCodeAt(3) is:" +
str.charCodeAt(3));
      document.write("<br />str.charCodeAt(4) is:" +
str.charCodeAt(4));
      document.write("<br />str.charCodeAt(5) is:" +
str.charCodeAt(5));
    </script>
  </body>
</html>
```

Output

```
str.charCodeAt(0) is:84
str.charCodeAt(1) is:104
str.charCodeAt(2) is:105
str.charCodeAt(3) is:115
str.charCodeAt(4) is:32
str.charCodeAt(5) is:105
```

JavaScript String - concat() Method

Description

This method adds two or more strings and returns a new single string.

Syntax

Its syntax is as follows −

```
string.concat(string2, string3[, ..., stringN]);
```

Argument Details

string2...stringN − These are the strings to be concatenated.

Return Value

Returns a single concatenated string.

Example

```
<html>
  <head>
    <title>JavaScript String concat() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str1 = new String( "This is string one" );
      var str2 = new String( "This is string two" );
      var str3 = str1.concat( str2 );
      document.write("Concatenated String :" + str3);
    </script>
  </body>
</html>
```

Output

```
Concatenated String :This is string oneThis is string two.
```

JavaScript String - indexOf() Method

Description

This method returns the index within the calling String object of the first occurrence of the specified value, starting the search at **fromIndex** or -1 if the value is not found.

Syntax

Use the following syntax to use the indexOf() method.

```
string.indexOf(searchValue[, fromIndex])
```

Argument Details

- **searchValue** − A string representing the value to search for.
- **fromIndex** − The location within the calling string to start the search from. It can be any integer between 0 and the length of the string. The default value is 0.

Return Value

Returns the index of the found occurrence, otherwise -1 if not found.

Example

```
<html>
  <head>
    <title>JavaScript String indexOf() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str1 = new String( "This is string one" );
      var index = str1.indexOf( "string" );
      document.write("indexOf found String :" + index );

      document.write("<br />");
      var index = str1.indexOf( "one" );
      document.write("indexOf found String :" + index );
    </script>
  </body>
</html>
```

Output

```
indexOf found String :8
indexOf found String :15
```

JavaScript String - lastIndexOf() Method

Description

This method returns the index within the calling String object of the last occurrence of the specified value, starting the search at **fromIndex** or -1 if the value is not found.

Syntax

Its syntax is as follows −

```
string.lastIndexOf(searchValue[, fromIndex])
```

Argument Details

- **searchValue** − A string representing the value to search for.
- **fromIndex** − The location within the calling string to start the search from. It can be any integer between 0 and the length of the string. The default value is 0.

Return Value

Returns the index of the last found occurrence, otherwise -1 if not found.

Example

Try the following example.

Live Demo

```
<html>
  <head>
    <title>JavaScript String lastIndexOf() Method</title>
```

```
  </head>

  <body>
    <script type = "text/javascript">
       var str1 = new String( "This is string one and again
string" );
       var index = str1.lastIndexOf( "string" );
       document.write("lastIndexOf found String :" + index );

       document.write("<br />");
       var index = str1.lastIndexOf( "one" );
       document.write("lastIndexOf found String :" + index );
    </script>
  </body>
</html>
```

Output

```
lastIndexOf found String :29
lastIndexOf found String :15
```

JavaScript String - localeCompare() Method

Description

This method returns a number indicating whether a reference string comes before or after or is the same as the given string in sorted order.

Syntax

The syntax of localeCompare() method is −

```
string.localeCompare( param )
```

Argument Details

param − A string to be compared with *string* object.

Return Value

- **0** − If the string matches 100%.
- **1** − no match, and the parameter value comes before the *string* object's value in the locale sort order
- **-1** −no match, and the parameter value comes after the *string* object's value in the local sort order

Example

```
<html>
  <head>
    <title>JavaScript String localeCompare() Method</title>
```

```
  </head>

  <body>
    <script type = "text/javascript">
      var str1 = new String( "This is beautiful string" );
      var index = str1.localeCompare( "XYZ" );
      document.write("localeCompare first :" + index );

      document.write("<br />" );
      var index = str1.localeCompare( "AbCD ?" );
      document.write("localeCompare second :" + index );
    </script>
  </body>
</html>
```

Output

```
localeCompare first :-1
localeCompare second :1
```

JavaScript String - match() Method

Description

This method is used to retrieve the matches when matching a string against a regular expression.

Syntax

Use the following syntax to use the match() method.

```
string.match( param )
```

Argument Details

param − A regular expression object.

Return Value

- If the regular expression does not include the **g flag,** it returns the same result as **regexp.exec(string)**.
- If the regular expression includes the **g flag,** the method returns an Array containing all the matches.

Example

```
<html>
  <head>
    <title>JavaScript String match() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = "For more information, see Chapter 3.4.5.1";
      var re = /(chapter \d+(\.\d)*)/i;
```

```
      var found = str.match( re );
      document.write(found );
   </script>
  </body>
</html>
```

Output

```
Chapter 3.4.5.1,Chapter 3.4.5.1,.1
```

JavaScript String - replace() Method

Description

This method finds a match between a regular expression and a string, and replaces the matched substring with a new substring.

The replacement string can include the following special replacement patterns –

Pattern	Inserts
$$	Inserts a "$".
$&	Inserts the matched substring.

$`	Inserts the portion of the string that precedes the matched substring.
$'	Inserts the portion of the string that follows the matched substring.
$n or $nn	Where **n** or **nn** are decimal digits, inserts the **n**th parenthesized submatch string, provided the first argument was a RegExp object.

Syntax

The syntax to use the replace() method is as follows −

```
string.replace(regexp/substr, newSubStr/function[, flags]);
```

Argument Details

- **regexp** − **A RegExp** object. The match is replaced by the return value of parameter #2.
- **substr** − A String that is to be replaced by **newSubStr**.
- **newSubStr** − The String that replaces the substring received from parameter #1.
- **function** − A function to be invoked to create the new substring.
- **flags** − A String containing any combination of the RegExp flags: **g** - global match, **i** - ignore case, **m** -

match over multiple lines. This parameter is only used if the first parameter is a string.

Return Value

It simply returns a new changed string.

Example

```
<html>
  <head>
    <title>JavaScript String replace() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var re = /apples/gi;
      var str = "Apples are round, and apples are juicy.";
      var newstr = str.replace(re, "oranges");
      document.write(newstr );
    </script>
  </body>
</html>
```

Output

```
oranges are round, and oranges are juicy.
```

JavaScript String - search() Method

Description

This method executes the search for a match between a regular expression and this String object.

Syntax

Its syntax is as follows −

```
string.search(regexp);
```

Argument Details

regexp − A regular expression object. If a non-RegExp object **obj** is passed, it is implicitly converted to a RegExp by using **new RegExp(obj)**.

Return Value

If successful, the search returns the index of the regular expression inside the string. Otherwise, it returns -1.

Example

```
<html>
  <head>
```

```
    <title>JavaScript String search() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var re = /apples/gi;
      var str = "Apples are round, and apples are juicy.";

      if ( str.search(re) == -1 ) {
        document.write("Does not contain Apples" );
      } else {
        document.write("Contains Apples" );
      }
    </script>
  </body>
</html>
```

Output

Contains Apples

JavaScript String - slice() Method

Description

This method extracts a section of a string and returns a new string.

Syntax

The syntax for slice() method is −

```
string.slice( beginslice [, endSlice] );
```

Argument Details

- **beginSlice** − The zero-based index at which to begin extraction.
- **endSlice** − The zero-based index at which to end extraction. If omitted, slice extracts to the end of the string.

Return Value

If successful, slice returns the index of the regular expression inside the string. Otherwise, it returns -1.

Example

```
<html>
  <head>
    <title>JavaScript String slice() Method</title>
```

```
  </head>

  <body>
    <script type = "text/javascript">
      var str = "Apples are round, and apples are juicy.";
      var sliced = str.slice(3, -2);
      document.write( sliced );
    </script>
  </body>
</html>
```

Output

```
les are round, and apples are juic
```

JavaScript String - split() Method

Description

This method splits a String object into an array of strings by separating the string into substrings.

Syntax

Its syntax is as follows −

```
string.split([separator][, limit]);
```

Argument Details

- **separator** − Specifies the character to use for separating the string. If *separator* is omitted, the array returned contains one element consisting of the entire string.
- **limit** − Integer specifying a limit on the number of splits to be found.

Return Value

The split method returns the new array. Also, when the string is empty, split returns an array containing one empty string, rather than an empty array.

Example

```
<html>
  <head>
    <title>JavaScript String split() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = "Apples are round, and apples are juicy.";
      var splitted = str.split(" ", 3);
      document.write( splitted );
    </script>
```

```
  </body>
</html>
```

Output

```
Apples,are,round,
```

JavaScript String - substr() Method

Description

This method returns the characters in a string beginning at the specified location through the specified number of characters.

Syntax

The syntax to use substr() is as follows −

```
string.substr(start[, length]);
```

Argument Details

- **start** − Location at which to start extracting characters (an integer between 0 and one less than the length of the string).
- **length** − The number of characters to extract.

Note − If start is negative, substr uses it as a character index from the end of the string.

Return Value

The substr() method returns the new sub-string based on given parameters.

Example

```
<html>
  <head>
    <title>JavaScript String substr() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = "Apples are round, and apples are juicy.";
      document.write("(1,2): "    + str.substr(1,2));
      document.write("<br />(-2,2): "   + str.substr(-2,2));
      document.write("<br />(1): "      + str.substr(1));
      document.write("<br />(-20, 2): " + str.substr(-20,2));
      document.write("<br />(20, 2): "  + str.substr(20,2));
    </script>
  </body>
</html>
```

Output

```
(1,2): pp
```

```
(-2,2): y.
(1): pples are round, and apples are juicy.
(-20, 2): nd
(20, 2): d
```

JavaScript String - substring() Method

Description

This method returns a subset of a String object.

Syntax

The syntax to use substr() is as follows −

```
string.substring(indexA, [indexB])
```

Argument Details

- **indexA** − An integer between 0 and one less than the length of the string.
- **indexB** − (optional) An integer between 0 and the length of the string.

Return Value

The substring method returns the new sub-string based on given parameters.

Example

```
<html>
  <head>
    <title>JavaScript String substring() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = "Apples are round, and apples are juicy.";
      document.write("(1,2): "    + str.substring(1,2));
      document.write("<br />(0,10): "   + str.substring(0,
10));
      document.write("<br />(5): "      + str.substring(5));
    </script>
  </body>
</html>
```

Output

```
(1,2): p
(0,10): Apples are
(5): s are round, and apples are juicy.
```

JavaScript String - toLowerCase() Method

Description

This method returns the calling string value converted to lowercase.

Syntax

Its syntax is as follows −

```
string.toLowerCase( )
```

Return Value

Returns the calling string value converted to lowercase.

Example

```
<html>
  <head>
    <title>JavaScript String toLowerCase() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = "Apples are round, and Apples are Juicy.";
      document.write(str.toLowerCase( ));
```

```
    </script>
  </body>
</html>
```

Output

```
apples are round, and apples are juicy.
```

JavaScript String - toString() Method

Description

This method returns a string representing the specified object.

Syntax

Its syntax is as follows −

```
string.toString( )
```

Return Value

Returns a string representing the specified object.

Example

```
<html>
  <head>
    <title>JavaScript String toString() Method</title>
```

```
  </head>

  <body>
    <script type = "text/javascript">
      var str = "Apples are round, and Apples are Juicy.";
      document.write(str.toString( ));
    </script>
  </body>
</html>
```

Output

```
Apples are round, and Apples are Juicy.
```

JavaScript String - toUpperCase() Method

Description

This method returns the calling string value converted to uppercase.

Syntax

Its syntax is as follows −

```
string.toUpperCase( )
```

Return Value

Returns a string representing the specified object.

Example

```
<html>
  <head>
    <title>JavaScript String toUpperCase() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = "Apples are round, and Apples are Juicy.";
      document.write(str.toUpperCase( ));
    </script>
  </body>
</html>
```

Output

```
APPLES ARE ROUND, AND APPLES ARE JUICY.
```

JavaScript String - valueOf() Method

Description

This method returns the primitive value of a String object.

Syntax

Its syntax is as follows −

```
string.valueOf( )
```

Return Value

Returns the primitive value of a String object.

Example

```
<html>
  <head>
    <title>JavaScript String valueOf() Method</title>
  </head>

  <body>
    <script type = "text/javascript">
      var str = new String("Hello world");
      document.write(str.valueOf( ));
    </script>
  </body>
```

```
</html>
```

Output

```
Hello world
```