

[Technologies](#) ▼[References & Guides](#) ▼[Feedback](#) ▼[Sign in](#) 

Template literals

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them. They were called "template strings" in prior editions of the ES2015 specification.

Syntax

```
`string text`

`string text line 1
string text line 2`

`string text ${expression} string text`

tag `string text ${expression} string text`
```

Description

Template literals are enclosed by the back-tick (```) ([↗ grave accent](#)) character instead of double or single quotes. Template literals can contain placeholders. These are indicated by the dollar sign and curly braces (`${expression}`). The expressions in the placeholders and the text between them get passed to a function. The default function just concatenates the parts into a single string. If there is an expression preceding the template literal (tag here), this is called a "tagged template". In that case, the tag expression (usually a function) gets called with the processed template literal, which you can then manipulate before outputting. To escape a back-tick in a template literal, put a backslash `\` before the back-tick.

```
1 | `\` == ` ` // --> true
```

Multi-line strings

Any newline characters inserted in the source are part of the template literal. Using normal strings, you would have to use the following syntax in order to get multi-line strings:

```
1 | console.log('string text line 1\n' +  
2 | 'string text line 2');  
3 | // "string text line 1  
4 | // string text line 2"
```

To get the same effect using template literals, you can now write:

```
1 | console.log(`string text line 1  
2 | string text line 2`);  
3 | // "string text line 1  
4 | // string text line 2"
```

Expression interpolation

In order to embed expressions within normal strings, you would use the following syntax:

```
1 | var a = 5;  
2 | var b = 10;  
3 | console.log('Fifteen is ' + (a + b) + ' and\nnot ' + (2 * a + b) + '.');  
4 | // "Fifteen is 15 and  
5 | // not 20."
```

Now, with template literals, you are able to make use of the syntactic sugar making substitutions like this more readable:

```
1 | var a = 5;
2 | var b = 10;
3 | console.log(`Fifteen is ${a + b} and
4 | not ${2 * a + b}.`);
5 | // "Fifteen is 15 and
6 | // not 20."
```

Nesting templates

In certain times, nesting a template is the easiest and perhaps more readable way to have configurable strings. Within a backticked template it is simple to allow inner backticks simply by using them inside a placeholder `${ }` within the template. For instance, if condition `a` is true: then return this templated literal.

In ES5:

```
1 | var classes = 'header'
2 | classes += (isLargeScreen() ?
3 |     '' : item.isCollapsed ?
4 |     ' icon-expander' : ' icon-collapser');
```

In ES2015 with template literals and without nesting:

```
1 | const classes = `header ${ isLargeScreen() ? '' :
2 |     (item.isCollapsed ? 'icon-expander' : 'icon-collapser') }`;
```

In ES2015 with nested template literals:

```
1 | const classes = `header ${ isLargeScreen() ? '' :
2 |     `icon-${item.isCollapsed ? 'expander' : 'collapser'}` }`;
```

Tagged templates

A more advanced form of template literals are *tagged* templates. Tags allow you to parse template literals with a function. The first argument of a tag function contains an array of string values. The remaining arguments are related to the expressions. In the end, your function can return your manipulated string (or it can return something completely different as described in the next example). The name of the function used for the tag can be named whatever you want.

```
1  var person = 'Mike';
2  var age = 28;
3
4  function myTag(strings, personExp, ageExp) {
5
6      var str0 = strings[0]; // "that "
7      var str1 = strings[1]; // " is a "
8
9      // There is technically a string after
10     // the final expression (in our example),
11     // but it is empty (""), so disregard.
12     // var str2 = strings[2];
13
14     var ageStr;
15     if (ageExp > 99){
16         ageStr = 'centenarian';
17     } else {
18         ageStr = 'youngster';
19     }
20
21     return str0 + personExp + str1 + ageStr;
22
23 }
24
25 var output = myTag`that ${ person } is a ${ age }`;
26
27 console.log(output);
28 // that Mike is a youngster
```

Tag functions don't need to return a string, as shown in the following example.

```
1  function template(strings, ...keys) {
2      return (function( values ) {
```

```
2   return (function(...values) {
3       var dict = values[values.length - 1] || {};
4       var result = [strings[0]];
5       keys.forEach(function(key, i) {
6           var value = Number.isInteger(key) ? values[key] : dict[key];
7           result.push(value, strings[i + 1]);
8       });
9       return result.join('');
10  });
11  }
12
13  var t1Closure = template`${0}${1}${0}!`;
14  t1Closure('Y', 'A'); // "YAY!"
15  var t2Closure = template`${0} ${'foo'}!`;
16  t2Closure('Hello', {foo: 'World'}); // "Hello World!"
```

Raw strings

The special `raw` property, available on the first function argument of tagged templates, allows you to access the raw strings as they were entered, without processing escape sequences.

```
1  function tag(strings) {
2      console.log(strings.raw[0]);
3  }
4
5  tag`string text line 1 \n string text line 2`;
6  // logs "string text line 1 \n string text line 2" ,
7  // including the two characters '\' and 'n'
```

In addition, the `String.raw()` method exists to create raw strings just like the default template function and string concatenation would create.

```
1  var str = String.raw`Hi\n${2+3}!`;
2  // "Hi\n5!"
3
4  str.length;
5  // 6
6
7  str.split('').join(',');
8  // "H,i,\,n,5,!"
```

Tagged templates and escape sequences

ES2016 behavior

As of ECMAScript 2016, tagged templates conform to the rules of the following escape sequences:

- Unicode escapes started by `"\u"`, for example `\u00A9`
- Unicode code point escapes indicated by `"\u{"`, for example `\u{2F804}`
- Hexadecimal escapes started by `"\x"`, for example `\xA9`
- Octal literal escapes started by `"\"` and (a) digit(s), for example `\251`

This means that a tagged template like the following is problematic, because, per ECMAScript grammar, a parser looks for valid Unicode escape sequences, but finds malformed syntax:

```
1 | latex`\unicode`  
2 | // Throws in older ECMAScript versions (ES2016 and earlier)  
3 | // SyntaxError: malformed Unicode character escape sequence
```

ES2018 revision of illegal escape sequences

Tagged templates should allow the embedding of languages (for example [DSLs](#), or [LaTeX](#)), where other escapes sequences are common. The ECMAScript proposal [Template Literal Revision](#) (stage 4, to be integrated in the ECMAScript 2018 standard) removes the syntax restriction of ECMAScript escape sequences from tagged templates.

However, illegal escape sequences must still be represented in the “cooked” representation. They will show up as `undefined` element in the “cooked” array:



```
1 | function latex(str) {  
2 |   return { "cooked": str[0], "raw": str.raw[0] }  
3 | }  
4 |  
5 | latex`\unicode`  
6 |  
7 | // { cooked: undefined, raw: "\\unicode" }
```













Specifications

Specification	Status	Comment
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Template Literals' in that specification.	ST Standard	Initial definition. Defined in several section of the specification: Template Literals , Tagged Templates
ECMAScript Latest Draft (ECMA-262) The definition of 'Template Literals' in that specification.	D Draft	Defined in several section of the specification: Template Literals , Tagged Templates
Template Literal Revision	Stage 4 draft	Drops escape sequence restriction from tagged templates

Browser compatibility

New compatibility tables are in beta ▼

Basic support		
		41
		12
		34
		No
		28
		9

 	41
 	41
 	12
 	34
 	28
 	?
 	4.0
	Yes

Escape sequences allowed in tagged template literals

 	No
 	No
 	53
 	No
 	No
 	No
 	No
 	No
 	No
 	53
 	No
 	No
 	No
	No



Full support



No support



Compatibility unknown



Experimental. Expect behavior to change in the future.

See also

- `String`
 - `String.raw()`
 - Lexical grammar
 - [Template-like strings in ES3 compatible syntax](#)
 - ["ES6 in Depth: Template strings" on hacks.mozilla.org](#)
-