JavaScript String Methods

Search, extract and edit.

01 length, [], at() & charAt()

```
const str = 'Hello';
str.length;      // 5
str[0];          // 'H'
str.at(-1);      // 'o'
str[-1];         // undefined
str.at(9);       // undefined
str.charAt(9);   // ''
```

at() (ES2022) counts negative indexes from the end. Out of range, [] and at() give undefined and charAt() gives ''. length counts UTF-16 code units, not characters (16).

02 indexOf, lastIndexOf & search

```
const str = 'Hello World, Hello Universe';
str.indexOf('World');      // 6
str.indexOf('Hello', 1);   // 13
str.indexOf('hello');      // -1
str.lastIndexOf('o');      // 17
str.search(/world/i);      // 6
```

Each returns an index, or -1 for no match. indexOf and lastIndexOf are case-sensitive and take a start position; search takes a regex and ignores its g flag.

03 includes, startsWith & endsWith

```
const url = 'https://example.com/api';
url.includes('example');        // true
url.startsWith('https');        // true
url.startsWith('example', 8);   // true
url.endsWith('.com', 19);       // true
url.includes(/api/);            // TypeError
```

Case-sensitive. The second argument is the start position, or for endsWith the length to check. A regex argument throws.

04 slice(), substring() & substr()

```
const str = 'Hello World';
str.slice(0, 5);       // 'Hello'
str.slice(-5, -1);     // 'Worl'
str.slice(5, 2);       // ''
str.substring(5, 2);   // 'llo'
str.substring(-3);     // 'Hello World'
str.substr(6, 3);      // 'Wor' (deprecated)
```

slice counts negative indexes from the end; substring treats them as 0 and swaps its arguments if start is larger. substr(start, length) is deprecated (Annex B).

05 replace() & replaceAll()

```
const s = 'cat cat';
s.replace('cat', 'dog');      // 'dog cat'
s.replaceAll('cat', 'dog');   // 'dog dog'
s.replace(/cat/g, 'dog');     // 'dog dog'
s.replaceAll(/cat/, 'dog');   // TypeError
s.replace(/a/g, a => a.toUpperCase());
// 'cAt cAt'
```

A string pattern replaces the first match only. replaceAll (ES2021) replaces all and throws on a regex without g. In the replacement, $1 is a group and $& the match.

06 split() & join()

```
'a,b,c'.split(',');      // ['a', 'b', 'c']
'a,b,c'.split(',', 2);   // ['a', 'b']
'a  b'.split(' ');       // ['a', '', 'b']
'a  b'.split(/\s+/);     // ['a', 'b']
['a', 'b'].join('-');    // 'a-b'
```

split('') cuts by UTF-16 code unit and breaks emoji: use [...str] (16). join() with no argument uses a comma.

07 trim(), trimStart() & trimEnd()

```
const s = '  hi  ';
s.trim();                  // 'hi'
s.trimStart();             // 'hi  '
s.trimEnd();               // '  hi'
'\t hi \r\n'.trim();       // 'hi'
'\u200Bhi'.trim().length;  // 3
```

Removes spaces, tabs and line breaks from the ends, also the no-break space and BOM, but not U+200B (zero-width space). trimStart and trimEnd are ES2019.

08 padStart() & padEnd()

```
'5'.padStart(3, '0');       // '005'
'5'.padEnd(3, '0');         // '500'
'Hi'.padStart(5);           // '   Hi'
'Hi'.padStart(7, 'abc');    // 'abcabHi'
'12345'.padStart(3, '0');   // '12345'
'5'.padStart(3, '');        // '5'
```

ES2017. The first argument is the final length. The pad string repeats and is cut to fit; the default is a space.

09 Changing case

```
'Hello'.toUpperCase();          // 'HELLO'
'Hello'.toLowerCase();          // 'hello'
'straße'.toUpperCase();         // 'STRASSE'
'i'.toLocaleUpperCase('tr');    // 'İ'
const cap = s =>
  s.charAt(0).toUpperCase() + s.slice(1);
cap('hello');                   // 'Hello'
```

toUpperCase and toLowerCase ignore the locale; the toLocale versions apply rules such as the Turkish dotted and dotless i. The length can change.

DevSheets

JavaScript String Methods

Templates, regex and Unicode.

10 repeat(), concat() & +

```
'ab'.repeat(3);        // 'ababab'
'ab'.repeat(0);        // ''
'ab'.repeat(-1);       // RangeError
'a'.concat('b', 1);    // 'ab1'
'a' + 1 + 2;           // 'a12'
1 + 2 + 'a';           // '3a'
```

concat joins like +, which most code uses instead. + runs left to right, so numbers before the first string are added.

11 Template literals

```
const name = 'Maya', n = 2;
`Hi, ${name}!`;       // 'Hi, Maya!'
`Total: ${n * 5}`;    // 'Total: 10'
`${{ id: 1 }}`;       // '[object Object]'
const text = `line 1
line 2`;              // 'line 1\nline 2'
```

Any expression can go in ${}; its value is converted to a string. Line breaks and indentation inside the backticks are kept. Write \` and \${ for a literal backtick and ${.

12 Tagged templates & String.raw

```
const mark = (strings, ...vals) =>
  strings.reduce((out, s, i) =>
    `${out}<b>${vals[i - 1]}</b>${s}`);
mark`Hi ${'Al'}, you are ${30}`;
// 'Hi <b>Al</b>, you are <b>30</b>'
`C:\temp\new`.length;            // 9
String.raw`C:\temp\new`.length;  // 11
```

A tag function gets the text parts and the ${} values. String.raw keeps backslashes as typed (Windows paths, regex sources) but still fills in ${} and cannot end with a backslash.

13 Methods that take a regex

match(re)	First match, or all with g
matchAll(re)	Iterator of all; needs g
replace(re, to)	First match, or all with g
replaceAll(re, to)	All matches; needs g
search(re)	Index of first match or -1
split(re)	Cut at every match

match, matchAll and search turn a string into a regex, so '.' matches any character; replace and split do not.

14 match() & matchAll()

```
const s = 'x=1, y=22';
s.match(/\d+/)[0];        // '1'
s.match(/\d+/g);          // ['1', '22']
s.match(/z/);             // null
const all = [...s.matchAll(/(\w)=(\d+)/g)];
all.map(m => m[2]);       // ['1', '22']
```

With g, match drops groups; no match is null, not []. matchAll (ES2020) keeps groups and index and throws without g.

15 localeCompare() & Intl.Collator

```
'ä'.localeCompare('z', 'de');   // -1
'ä'.localeCompare('z', 'sv');   // 1
const col = new Intl.Collator('en',
  { numeric: true, sensitivity: 'base' });
col.compare('a', 'A');          // 0
['a10', 'a2'].sort(col.compare);
// ['a2', 'a10']
```

Returns a negative number, 0 or a positive one: test < 0, not === -1. localeCompare takes the same locale and options.

16 Code units & code points

```
const s = String.fromCodePoint(0x1F600);
s === '\u{1F600}';     // true (one emoji)
s.length;              // 2
s.charCodeAt(0);       // 55357
s.codePointAt(0);      // 128512
s.split('').length;    // 2
[...s].length;         // 1
```

length, [] and split('') count UTF-16 code units, and an emoji takes two. Spread, for...of and Array.from go by code point; Intl.Segmenter by visible character.

17 normalize() & isWellFormed()

```
// é as one code point, then as e + accent
const a = '\u00E9', b = 'e\u0301';
a === b;                              // false
a === b.normalize();                  // true
const bad = 'a\uD83D';
bad.isWellFormed();                   // false
bad.toWellFormed() === 'a\uFFFD';     // true
```

The default form is NFC (composed); 'NFD' decomposes. Normalize before comparing. isWellFormed and toWellFormed (ES2024) find and fix lone surrogates.

18 Immutability & deprecated methods

s[0] = 'x'	Ignored; TypeError in strict mode
substr()	Deprecated: use slice()
trimLeft, trimRight	Use trimStart, trimEnd
anchor, bold, link	Deprecated HTML wrappers

No method changes a string: each returns a new one, so keep the result (s = s.trim()).