Строки

JavaScript любые текстовые данные являются строками. Внутренний формат для строк — всегда UTF-16, вне зависимости от кодировки страницы. В JavaScript есть разные типы кавычек. Строку можно создать с помощью одинарных, двойных либо обратных кавычек. Одинарные и двойные кавычки работают, по сути, одинаково, а если использовать обратные кавычки, то в такую строку мы сможем вставлять произвольные выражения, обернув их в ${…}:

```
alert(`1 + 2 = ${sum(1, 2)}.`); // 1 + 2 = 3
```

Ещё одно преимущество обратных кавычек — они могут занимать более одной строки. Многострочные строки также можно создавать с помощью одинарных и двойных кавычек, используя так называемый «символ перевода строки», который записывается как \n:

```
let guestList = "Guests:\n * John\n * Pete\n * Mary";
```

Есть и другие, реже используемые спецсимволы. Вот список:

Символ	Описание
`\n`	Перевод строки
`\r`	Возврат каретки: самостоятельно не используется. В текстовых файлах Windows для перевода строки используется комбинация символов `\r\n`.
`\'`, `\"`	Кавычки
`\\`	Обратный слеш
`\t`	Знак табуляции
`\b`, `\f`, `\v`	Backspace, Form Feed и Vertical Tab — оставлены для обратной совместимости, сейчас не используются.
`\xXX`	Символ с шестнадцатеричным юникодным кодом `XX`, например, `'\x7A'` — же самое, что `'z'`.
`\uXXXX`	Символ в кодировке UTF-16 с шестнадцатеричным кодом `XXXX`, например, `\u00A9` — юникодное представление знака копирайта, `©`. Код должен состоять ровно из 4 шестнадцатеричных цифр.
`\u{X…XXXXXX}` (от 1 до 6 шестнадцатеричных цифр)	Символ в кодировке UTF-32 с шестнадцатеричным кодом от U+0000 до U+10FFFF. Некоторые редкие символы кодируются двумя 16-битными словами и занимают 4 байта. Так можно вставлять символы с длинным кодом.

Примеры с Юникодом:

```
// ©
alert( "\u00A9" );
// Длинные юникодные коды
// 😍, лицо с улыбкой и глазами в форме сердец
alert( "\u{1F60D}" );
```

str.length —свойство, длина строки. Добавлять скобки не нужно!

Доступ к символам

str[pos] – символ, который занимает позицию pos. Можно также str.charAt(pos)

Разница между ними: если символ с такой позицией отсутствует, тогда [] вернёт undefined, а charAt — пустую строк

Содержимое строки в JavaScript нельзя изменить. Нельзя взять символ посередине и заменить его. Как только строка создана — она такая навсегда. Можно создать новую строку и записать её в ту же самую переменную вместо старой.

```
let str = 'Hi';
```

```
str = 'h' + str[1]; // заменяем строку
```

Изменение регистра

Методы toLowerCase() и toUpperCase() меняют регистр символов:

```
alert( 'Interface'.toUpperCase() ); // INTERFACE
alert( 'Interface'.toLowerCase() ); // interface
```

Если мы захотим перевести в нижний регистр какой-то конкретный символ:

```
alert( 'Interface'[0].toLowerCase() ); // 'i'
```

Поиск, получение подстроки

str.indexOf(substr, pos) - Метод ищет подстроку substr в строке str, начиная с позиции pos, и возвращает первую позицию, на которой располагается совпадение, либо -1 при отсутствии совпадений. Н-р:

```
let str = 'Widget with id';
alert( str.indexOf('widget') ); // -1, совпадений нет, поиск чувствителен к регистру
alert( str.indexOf("id") ); // 1
```

str.lastIndexOf(substr, position) - ищет с конца строки к её началу.

Проверка на наличие совпадений (проверяется на -1):

```
let str = "Widget with id";

if (str.indexOf("Widget") != -1) {
   alert("Совпадение есть"); // теперь работает
}
```

Существует старый трюк с использованием побитового оператора НЕ — ~. Он преобразует число в 32-разрядное целое со знаком (signed 32-bit integer). Дробная часть, в случае, если она присутствует, отбрасывается. Затем все биты числа инвертируются. На практике это означает простую вещь: для 32-разрядных целых чисел значение **~n равно -(n+1).**

```
alert( ~1 ); // -2, то же, что -(1+1)
alert( ~0 ); // -1, то же, что -(0+1)
```

Это иногда применяют, чтобы сделать проверку indexOf компактнее:

```
let str = "Widget";
if (~str.indexOf("Widget")) {
  alert( 'Совпадение есть' ); // работает
}
```

Более современный метод str.includes(substr, pos) возвращает true, если в строке str есть подстрока substr, либо false, если нет.

```
alert( "Widget with id".includes("Widget") ); // true
```

Метод includes() выполняет поиск с учетом регистра, чтобы определить, можно ли найти одну строку в другой строке, возвращая true или false в зависимости от ситуации.

```
console.log("name.includes('hn D')", name.includes('hn D'));    // true
```

Методы str.startsWith и str.endsWith проверяют, соответственно, начинается ли и заканчивается ли строка определённой строкой:

```
alert( "Widget".startsWith("Wid") ); // true, "Wid" — начало "Widget"
alert( "Widget".endsWith("get") ); // true, "get" — окончание "Widget"
```

str.slice(start [, end]) - возвращает часть строки от start до (не включая) end. Если аргумент end отсутствует, slice возвращает символы до конца строки. Также для start/end можно задавать отрицательные значения. Это означает, что позиция определена как заданное количество символов с конца строки.

str.substring(start [, end]) - возвращает часть строки между start и end. Это — почти то же, что и slice, но можно задавать start больше end. Отрицательные значения substring, в отличие от slice, не поддерживает, они интерпретируются как 0.

str.substr(start [, length])
Возвращает часть строки от start длины length. Значение первого аргумента может быть отрицательным, тогда позиция определяется с конца.

Другие методы

str.trim() — убирает пробелы в начале и конце строки.
str.repeat(n) — повторяет строку n раз.
Также есть методы для поиска и замены с использованием **регулярных выражений**.

str.split(delim)– преобразует строку в массив

Сравнение строк

Строки сравниваются посимвольно по их кодам. Больший код — больший символ. Все строчные буквы идут после заглавных, так как их коды больше. Некоторые буквы, такие как Ö, вообще находятся вне основного алфавита. У этой буквы код больше, чем у любой буквы от a до z.

Алгоритм сравнения двух строк довольно прост:

1. Сначала сравниваются первые символы строк.
2. Если первый символ первой строки больше(меньше), чем первый символ второй, то первая строка больше(меньше) второй.Сравнение завершено.
3. Если первые символы равны, то таким же образом сравниваются уже вторые символы строк.
4. Сравнение продолжается, пока не закончится одна из строк.
5. Если обе строки заканчиваются одновременно, то они равны.Иначе, большей считается более длинная строка.

str.codePointAt(pos) - возвращает код для символа, находящегося на позиции pos:

```
// одна и та же буква в нижнем и верхнем регистре будет иметь разные коды
alert( "z".codePointAt(0) ); // 122
alert( "Z".codePointAt(0) ); // 90
```

String.fromCodePoint(code) - создаёт символ по его коду code

```
alert( String.fromCodePoint(90) ); // Z
```

Также можно добавлять юникодные символы по их кодам, используя \u с шестнадцатеричным кодом символа:

```
// 90 — 5a в шестнадцатеричной системе счисления
alert( '\u005a' ); // Z
```

«Правильный» алгоритм сравнения строк сложнее, чем может показаться, так как разные языки используют разные алфавиты. Поэтому браузеру нужно знать, какой язык использовать для сравнения. К счастью, все современные браузеры (для IE10− нужна дополнительная библиотека Intl.JS) поддерживают стандарт ECMA 402, обеспечивающий правильное сравнение строк на разных языках с учётом их правил.Для этого есть соответствующий метод.

str.localeCompare(str2) возвращает число, которое показывает, какая строка больше в соответствии с правилами языка:

- Отрицательное число, если str меньше str2.
- Положительное число, если str больше str2.
- 0, если строки равны

Например:

```
alert( 'Österreich'.localeCompare('Zealand') ); // -1
```